

AD-A163 842

COMPUTER ASSISTED INSTRUCTION FOR THE 'C' PROGRAMMING
LANGUAGE ON THE ZEM. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. F M DENARCO

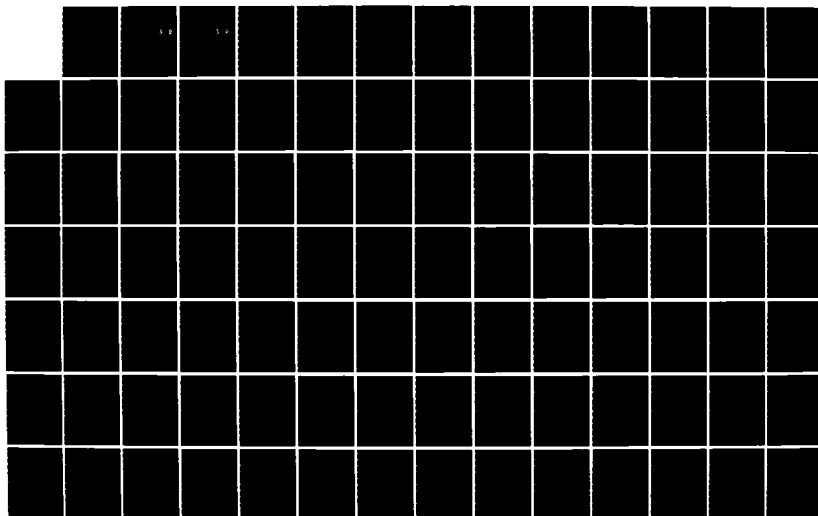
1/3

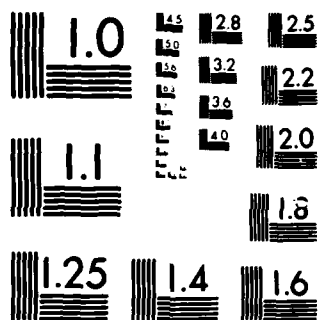
UNCLASSIFIED

DEC 85 AFIT/GCS/NA/85D-2

F/G 9/2

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

AD-A163 842



DTIC
ELECTE
FEB 11 1986
S D

COMPUTER ASSISTED INSTRUCTION FOR THE
"C" PROGRAMMING LANGUAGE ON THE
ZENITH Z-100 MICROCOMPUTER SYSTEM

THESIS

Frank W. DeMarco
Captain, USAF

AFIT/GCS/MA/85D-2

DISTRIBUTION STATEMENT A

Approved for public release;
Distribution Unlimited

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

86 2 10 019

DTIC FILE COPY

AFIT/GCS/MA/85

①

DTIC
ELECTE
FEB 11 1986
S D D

COMPUTER ASSISTED INSTRUCTION FOR THE
"C" PROGRAMMING LANGUAGE ON THE
ZENITH Z-100 MICROCOMPUTER SYSTEM

THESIS

Frank W. DeMarco
Captain, USAF

AFIT/GCS/MA/85D-2

Approved for public release; distribution unlimited

AFIT/GCS/MA/85D-2

COMPUTER ASSISTED INSTRUCTION FOR THE "C" PROGRAMMING
LANGUAGE ON THE ZENITH Z-100 MICROCOMPUTER SYSTEM

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology

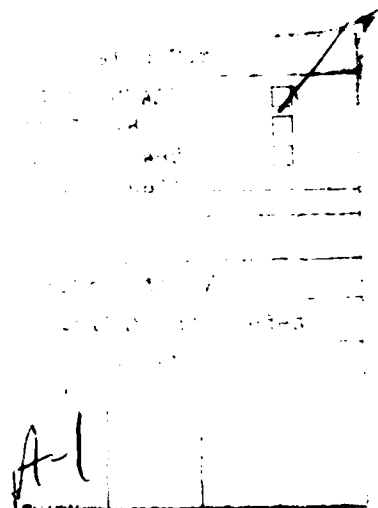
Air University

In Partial Fulfillment of the
Requirements for the Degree of
Master of Science

Frank W. DeMarco, B.S.

Captain, USAF

December 1985



Approved for public release; distribution unlimited

Preface

The purpose of this study was to develop a computer assisted instruction (CAI) program package for use on the Zenith Z-100 microcomputer system. The package is designed to give programming students introductory information on the "C" programming language. This programming package is to be used in the training programs managed by the Computer Assisted Instruction Plans Branch of the 3300 Technical Training Wing at Keesler AFB, Mississippi.

I would like to express my sincere thanks to my thesis advisor, Dr. Henry B. Potoczny, who gave me guidance and encouragement throughout my thesis effort. Thanks is also extended to Captain Patricia Lawlis, who as my thesis reader provided many constructive comments on improving this thesis. Grateful appreciation is also extended to the sponsor my thesis, the CAI Plans Branch at Keesler AFB, and in particular, Captain Glen A. Miller and Technical Sergeant Charles T. Neal, who provided help in the verification and validation of the programs and course material I developed.

Finally, I want to express special gratitude to my wife Anne and my children Crystal and Bryan. They have forfeited countless hours of time with me in order that I could complete my graduate work here at AFIT. Their patience, understanding, and devoted love gave me the strength I needed to overcome the many obstacles I encountered. I owe them a debt that will take a lifetime to repay.

Table of Contents

	Page
Preface	ii
List of Figures	v
Abstract	vi
I. Introduction	1-1
Background	1-1
Statement of Problem	1-2
Scope	1-3
Assumptions	1-3
General Approach	1-4
II. Methodology	2-1
The Aim of CAI	2-1
Advantages of CAI	2-1
Disadvantages of CAI	2-3
Development Considerations	2-4
Course Development Approach	2-5
III. Design Specification	3-1
General Description	3-1
CAI Program	3-1
Status Program	3-2
Statistics Program	3-3
IV. System Implementation	4-1
General Description	4-1
"C" Lessons Descriptions	4-1
CAI Program	4-3
Status Program	4-13
Statistics Program	4-15
V. Conclusions and Recommendations	5-1
General Comments	5-1
Suggestions for Further Study	5-1
Appendix A: Users Guide	A-1
Using Program "CAI"	A-1
Using Program "Student_Status"	A-1
Using Program "CAI_Statistics"	A-2

	Page
Appendix B: Program Listings	B-1
Program "CAI"	B-1
Program "STUDENT_STATUS"	B-34
Program "CAI_STATISTICS"	B-41
Appendix C: Files Used by Program "CAI"	C-1
File "INTRO"	C-1
File "MENU"	C-2
File "LESSON1"	C-3
File "LESSON2"	C-28
File "LESSON3"	C-51
File "LESSON4"	C-72
File "LESSON5"	C-92
File "LESSON6"	C-109
File "EXIT"	C-128
Vita	V-1

List of Figures

Figure	Page
4.1 CAI - Main	4-5
4.2 CAI - Query	4-6
4.3 CAI - StartLesson	4-7
4.4 CAI - ShowTopic	4-8
4.5 CAI - Tframe	4-9
4.6 CAI - Qframe	4-10
4.7 CAI - Mquestion	4-11
4.8 CAI - Pquestion	4-12
4.9 Student_Status - Main	4-14
4.10 CAI_Statistics - Main	4-16
4.11 CAI_Statistics - Display	4-17
4.12 CAI_Statistics - ShowStats	4-18

Abstract

The field known as "computer assisted instruction" or "CAI" as it is commonly called, has gained considerable interest and support since the advent of the microcomputer. More and more people, including those in supervisory positions are beginning to see the advantages, both cost and time, in having training available in the workplace. This study developed a training package for use on the Zenith 2-100 microcomputer. The package consists of six lessons and three programs. The six lessons cover various topics dealing with the "C" programming language. The objective of these lessons is to present an introduction to the "C" programming language. The three programs are written in the Pascal programming language and are used for the following functions:

1. Provide a means of displaying the lesson material.
2. Provide a means of checking student progress.
3. Provide a means of displaying course statistics.

Handwritten signature and date: 10/10/85

Computer Assisted Instruction
for the
"C" Programming Language
on the
Zenith Z-100 Microcomputer System

I. Introduction

Background

The use of Computer Assisted Instruction (CAI) to help in the training needs of the Services has increased with the introduction of microcomputer systems into the workplace. The development of CAI courses for use on these computer systems has been lagging behind the need for training on the new systems. The CAI development process involves a working knowledge of the system to present the material as well as a knowledge of the subject to be presented. The presentation of the developed CAI course is usually controlled by means of some type of presentation program. Manpower and time constraints may prohibit development of such a program and indicate the need to utilize a commercially developed authoring/presentation system.

The use of a commercial authoring system requires that a coursewriter learn that specific authoring system for use on a specific microcomputer system. The coursewriter can then devote his attention to the development of the course subject material. The subject topics that typically are identified as of primary importance include: word processing, data base management, spreadsheets, operating systems, and programming languages.

Statement of Problem

The problem to be solved is as follows: How can a computer assisted instruction (CAI) course be written and implemented to teach the "C" programming language on the Zenith Z-100 microcomputer system without the use of a commercial authoring/presentation system? The course will be of sufficient length to instruct the beginning student to a level that will allow him/her to program using the "C" programming language. The course subject will be broken into lesson topics which are made up of subsections of the lesson topic.

Each lesson will:

1. Give the student the ability to select between being shown the complete lesson or only reviewing certain parts.
2. Have the ability to sample student comprehension during lesson presentation by means of questions.
3. Have the ability to branch, at appropriate times, to other parts of the lesson.
4. Give the student a chance to review subsections before being tested on the lesson material.
5. Have the ability to test the student on the presented material after lesson completion.
6. Have the ability to allow for review of subsections before retesting (in the case of lesson failure).

In addition to the above, a record will be kept of student responses, both during the presentation of the lessons and the tests, for later statistical analysis and

display. This is done in order to be able to identify areas of the course that perhaps are not teaching the material as intended and/or are causing the student difficulties.

Scope

The scope of this thesis effort is to design, implement, test, and validate a CAI course for presenting information on the "C" programming language. The design phase will incorporate top down structured programming techniques.

Although it is not the primary purpose of this thesis, a method for developing the textual material and presenting that material will necessarily be created. This added benefit arises from the fact that no commercially available authoring/presentation software will be used. This opens up the possibility of developing other courses using programs written during this thesis effort.

The end result of this thesis is to develop a CAI course that will be acceptable to the sponsor at Keesler AFB, who will then distribute the course to all interested training managers throughout the Services.

Assumptions

It is assumed that the students who will use this CAI course will have a working knowledge of the operation of the Zenith Z-100 microcomputer system that this course is designed to run on. This Z-100 system is the standard system purchased by government contract through Zenith Data

Systems, namely, the 192K byte, two 5.25 inch disk drive system. Although it would be of some benefit, there isn't any requirement that students taking this CAI course have access to a "C" compiler.

General Approach

The first step in this endeavor is to do research into the techniques of teaching with a computer. The purpose here is to broaden the teaching base from which to build the overall course presentation. Once the methods of presenting the material are well in hand, the course material will be researched to establish a firm background from which to teach. The next step is to write the individual lessons and develop the program to present them. A program will then be developed to do the statistical analysis and display.

Following the research and development phase will be the implementation of the system. This phase will consist of putting all the pieces into a cohesive package that will accomplish the goal of the study, namely, use the Zenith Z-100 computer system to present a CAI course on the "C" programming language.

In order to ensure the development and implementation of a quality product, an extensive testing and validation system will be incorporated throughout the study. The ultimate test will come when the sponsor at Keesler AFB tests the course against their well-established standards.

II. Methodology

The Aim of CAI

The overall aim of Computer Assisted Instruction (CAI) is, naturally, to use a computer system to assist in the training of individuals in a given subject.

In its most common form, CAI is very similar to a programmed text. The subject material is presented to the student, questions are asked of the student, answers are evaluated, and a decision is made as to what material is shown next. If the questioning indicates that the student understands the material, new material is shown. If the student seems to be having trouble with a particular part of the lesson, a branch can be made to supplemental material to help the student understand. Other forms of CAI use simulation and/or emulation techniques. These methods of instruction are very useful when teaching a specific performance process but not for such things as computer programming. Since it is the intent of this thesis effort to teach a programming language, the method used will closely resemble that of programmed text.

Advantages of CAI

There are many advantages to CAI, the following are but a few of the more important ones: Standardization, time efficiency, availability, flexibility, modularity, and cost efficiency.

Each of these advantages contribute to the overall attractiveness of using CAI as a method of training. Standardization is accomplished by programming the subject material into the computer. In this way, each and every student who takes a given course will receive the same information. The unfortunate human flaw of a human teacher forgetting to mention some important detail is thus avoided.

The use of CAI can save time by allowing students to progress at their own rate. This is opposed to the alternative of locking them into a classroom setting and controlling the pace of the class as a whole. This leads us to another of the advantages, that of availability. Since the CAI course is conducted on a microcomputer system, the course is virtually available at all times. This means that many training requirements can be accomplished without the student having to leave his/her work area. Hence you have flexibility (another advantage) in scheduling training. Since the course is available at all times, training can be scheduled around work requirements.

The actual construction of a well developed CAI package should allow for the accessing of information in a rapid way. This usually calls for the development of sections of the course in small modules. In this way the student doesn't necessarily need to complete an entire course to get at the information that pertains to his/her job requirements. The CAI course developed in this thesis follows this modular course concept. The course is broken into lessons

which are each broken into topics.

The last advantage mentioned is that of cost effectiveness. By taking the previous advantages into consideration it is easy to see how using CAI can achieve a cost effective training program. Training can be conducted whenever workload requirements allow the student enough free time to take CAI lessons. The need for the student to travel to some other location for needed training can also be reduced.

Disadvantages of CAI

There are of course drawbacks to everything, and CAI is no exception. A few of the disadvantages follow: System availability, Uni-directional training, acceptability.

The availability of the computer system for training purposes can restrict the usefulness of the CAI system. The primary reasons for system nonavailability are: Operational requirements and maintenance downtime. An organizations operational requirements may be such that a computer system can not be spared in order to accomplish a training requirement in a timely manner. Obviously, if a computer system is down for maintenance, training can not be accomplished using that system.

The second disadvantage involves the inability of the students to ask questions of their trainer. This requires the student to concentrate hard on the presented material in order to ensure the required understanding. Since the student doesn't (usually) have the ability to query the compu-

ter on a point that may be causing him/her problems, the student must seek out someone who knows the subject in order to receive clarification. This of course is not all bad, since this will lead to better communication in the work place.

Lastly, CAI is not completely accepted by management personnel as an alternative to classroom instruction. Formal classroom instruction has been used for so long that many believe it to be the only effective means of accomplishing required training.

Development Considerations

There are several design considerations to take into account when coming up with a methodology for CAI course development. The first of these and perhaps the most important is the objective of what is to be taught. As stated before, the objective of this study is to create a means of presenting information on the "C" programming language on the Z-100 microcomputer system. The second consideration is the resources available for training. The necessary resources for taking the course developed in this study is any microcomputer system that runs under the MS-dos operating system. The primary system will be the Zenith Z-100. The third consideration is the teaching technique to be used. As stated earlier, this will most closely resemble a programmed text presentation. The last major development consideration is course validation. In addition to initial

development validation, the CAI package will provide a means of recording student progress and provide for statistical collection of student responses to all questions throughout the course. These capabilities will be described in detail later in the study.

Course Development Approach

The general approach to developing this CAI was to write a program which would keep track of as many as twenty students as well as present the material to the student in short topic sessions. The student has total control over which topics he/she views. The presentation program is written in the PASCAL programming language and is easy to modify in the case of any future enhancements. Two additional programs, both written in PASCAL, have been included in the package. The purpose of the first is to produce a report of the current student status for each registered student on a given student disk. The purpose of the second is to produce a report of the statistics collected on the student responses during course presentation. More detailed information can be found in Chapter 3.

III. Design Specification

General Description

The purpose of this computer assisted instruction (CAI) package is to provide a means of presenting introductory information on the "C" programming language. The material to be presented is to be stored in separate lesson files on one five and one quarter inch floppy disk that has been formatted using the MS-DOS "format" program. The lesson files are to be created using any text editor that will run under MS-DOS. The lessons are to be broken into topic sections that the student can complete in a relatively short period of time. Three programs will be provided with the CAI package and a description of these follows.

CAI Program

The main program contained in the CAI package is the one that will present the course material to the student. This program will read and display several files automatically in addition to reading and displaying the student's chosen subject material. This program will keep a record of student progress through the course as well as write to a statistical collection file to be used for future course validation and improvements.

Lesson Files. The main CAI program will have access to six lesson files which contain the course material. Each of these lesson files will contain introductory information for its particular lesson material. An associated menu is also

included for display, allowing the student to choose the topic material to be shown.

Other Files. In addition to the lesson files discussed above, there are five other files to be used by the main CAI program. First, there is to be a file that contains introductory comments to the student. This file will be kept short since it is to be seen each and every time the program is executed. Second, there is to be a main menu file that will allow the student to choose which of the six lessons they want to enter. This file will be restricted to one screen in size. Third, there is to be a file that contains program conclusion comments. This file will also be kept short and is only to serve as a means of assuring the student that they have correctly terminated the program. Fourth, there is to be a file that will store data on as many as twenty students. This file is where each student's progress through the course is to be kept and will be keyed on a unique student identification number. Lastly, there is to be a file that will store data for each question displayed during course presentation. This file will be used for ongoing course improvements.

Status Program

The status program will be provided for use by the training monitor. Its whole purpose is to provide a means of viewing each student's record in order to determine their status in the course. A report is to be generated giving

the unique student identification number and listing all lessons that have been successfully passed. The program will allow for the possibility of the training monitor having merged several student record files into one file. The program will also format the report for either screen display or hardcopy printout.

"STUDENT" File. The file to be read by the status program is to contain student identification numbers, student names, as well as lesson and topic status data. An active student file will contain at most twenty unique student records. As mentioned earlier, several student files may be merged into one student file prior to running the status program.

Statistics Program

A statistics program will be provided for use by the office of primary responsibility (OPR) at Keesler AFB. This program is to provide a means of analyzing the data collected on questions presented during each training session. A report is to be generated giving information such as lesson number, frame number, number of responses for each of the valid responses, number of right responses, number of wrong responses, percent of right responses, and percent of wrong responses. The results of the statistical analysis is to be displayed in either of two formats: screen display or hardcopy printout.

"STATS" File. The file to be read by the statistics program is to contain such items as lesson number, topic number, frame number, correct answer, and the student's response. Again, several of these files may be merged into one file prior to running the statistical program.

IV. System Implementation

General Description

The implementation of this computer assisted instruction (CAI) training package involved the development of lesson material covering six major subject areas. The development of these lessons was accomplished in conjunction with the development and validation of the three programs specified in chapter three. This chapter presents a brief description of each major component of the CAI training package. Copies of these components are provided in appendixes B and C.

"C" Lessons Descriptions

The following is a breakdown of the subject material as presented in the "C" CAI course:

Lesson One. Lesson one contains introductory information on the course and some general information on "C" programming. The lesson is broken into four subtopics and a lesson test. The first subtopic gives a short introduction to the overall course structure and some of the particulars used in the course. The second subtopic discusses the overall organization and structure of a typical C program. The third subtopic gives a description of the overall C programming environment covering such items as "compiling" and "linking". The forth subtopic states a problem to be solved and presents a solution to help introduce the student to C program statements.

Lesson Two. Lesson two contains information on variables, constants, operators, and expressions used in C programming. The lesson is broken into four subtopics and a lesson test. The first and second subtopics cover the declaration and use of variables and constants. The third and forth subtopics cover the use of the different operators and expressions in C programming.

Lesson Three. Lesson three contains information on program control statements used in C programming. The lesson is broken into four subtopics and a lesson test. The first subtopic gives descriptions of the structure and use of the "if" and "if-else" control statements and how to "nest" these statements along with a description of the "switch" control statement. The second subtopic discusses the structure and use of loop statements (while, for, and do-while). The third subtopic gives a description of the "break" and "continue" statements and how they are used. The forth subtopic gives a description of the "goto" statement and the use of "labels" within a C program.

Lesson Four. Lesson four contains information on arrays, pointers, and address arithmetic used in C programming. The lesson is broken into four subtopics and a lesson test. The first subtopic introduces the declaration, initialization, and use of arrays. The second subtopic introduces the declaration and use of pointers. The third and forth subtopics cover how to work with pointers and includes topics such as how pointers are passed to functions, how pointers

are used in conjunction with arrays, and how to use address arithmetic.

Lesson Five. Lesson five contains information on structures that are used in C programming. The lesson is broken into four subtopics and a lesson test. The first subtopic introduces the idea of structures and two methods of their declaration. The second subtopic describes the use of structures within structures and arrays of structures. The third subtopic describes how to use pointers in conjunction with structures. The forth subtopic describes how structures are passed between functions.

Lesson Six. Lesson six contains introductory information on input and output capabilities of the C language. The lesson is broken into four subtopics and a lesson test. The first subtopic gives a description of the use of the standard I/O functions "getchar" and "putchar". The second subtopic gives a description of the use of the standard input function "getline". The third subtopic gives a description and examples of the standard input function "scanf". The forth subtopic gives a description and examples of the standard output function "printf".

CAI Program

Program CAI is the program that is used to present the lesson material to the student. The program is designed to present any lesson material that is in the same format as the lessons developed in this thesis. Therefore, additional

courses may be written for presentation on the Zenith Z-100 by this program. The following is a breakdown and brief description of the program.

Structure Charts. The program is broken into a main program and 17 procedures, all of which are written in the Pascal programming language. Structure charts of this program are presented in Figures 4.1 thru 4.8 of this chapter.

Flow Description. The flow of this program follows a very logical structured path. The program begins by presenting an introductory message from file "INTRO". The student is next queried for their unique student identification number. A search is then made of file "STUDENT" (which has been read into memory) and if no match is found (a new student) the student is queried for their name and unique student identification number they wish to use from this point on. Next, the student is presented a menu of lessons from which to choose (file "MENU"). Once a selection is made, introductory information for the chosen lesson is displayed and another menu is presented giving the student a choice of lesson subtopics. Once the student chooses a lesson subtopic, the topic is read into memory and topic presentation begins. When the topic is completed, an update of the CAI statistical collection file as well as the students progress record file is made. The student is then returned to the subtopic selection menu, where if the student wishes, he/she may exit to the lesson selection menu, where if the student wishes, he/she may exit the program.

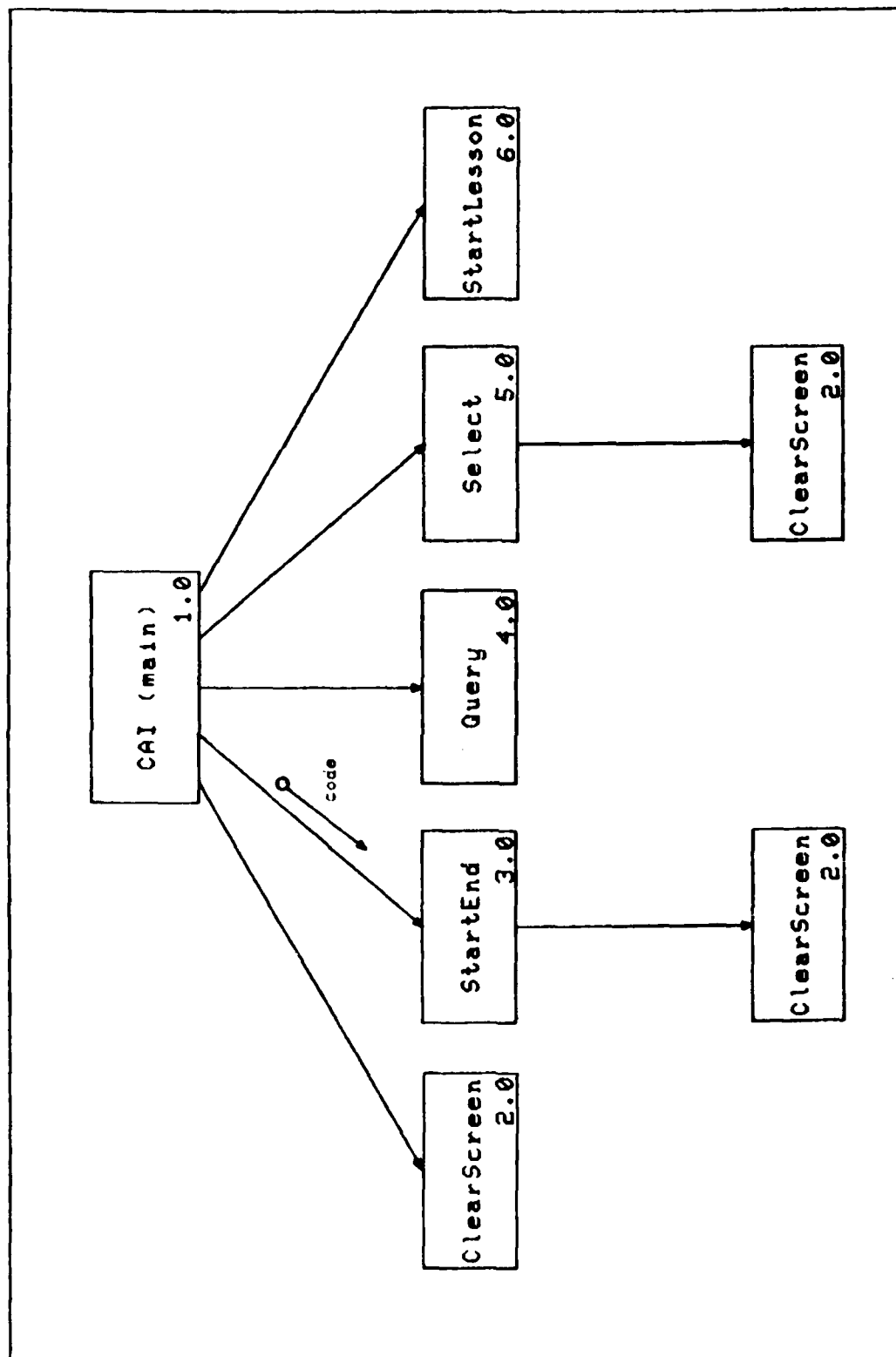


Figure 4.1 CAI - Main

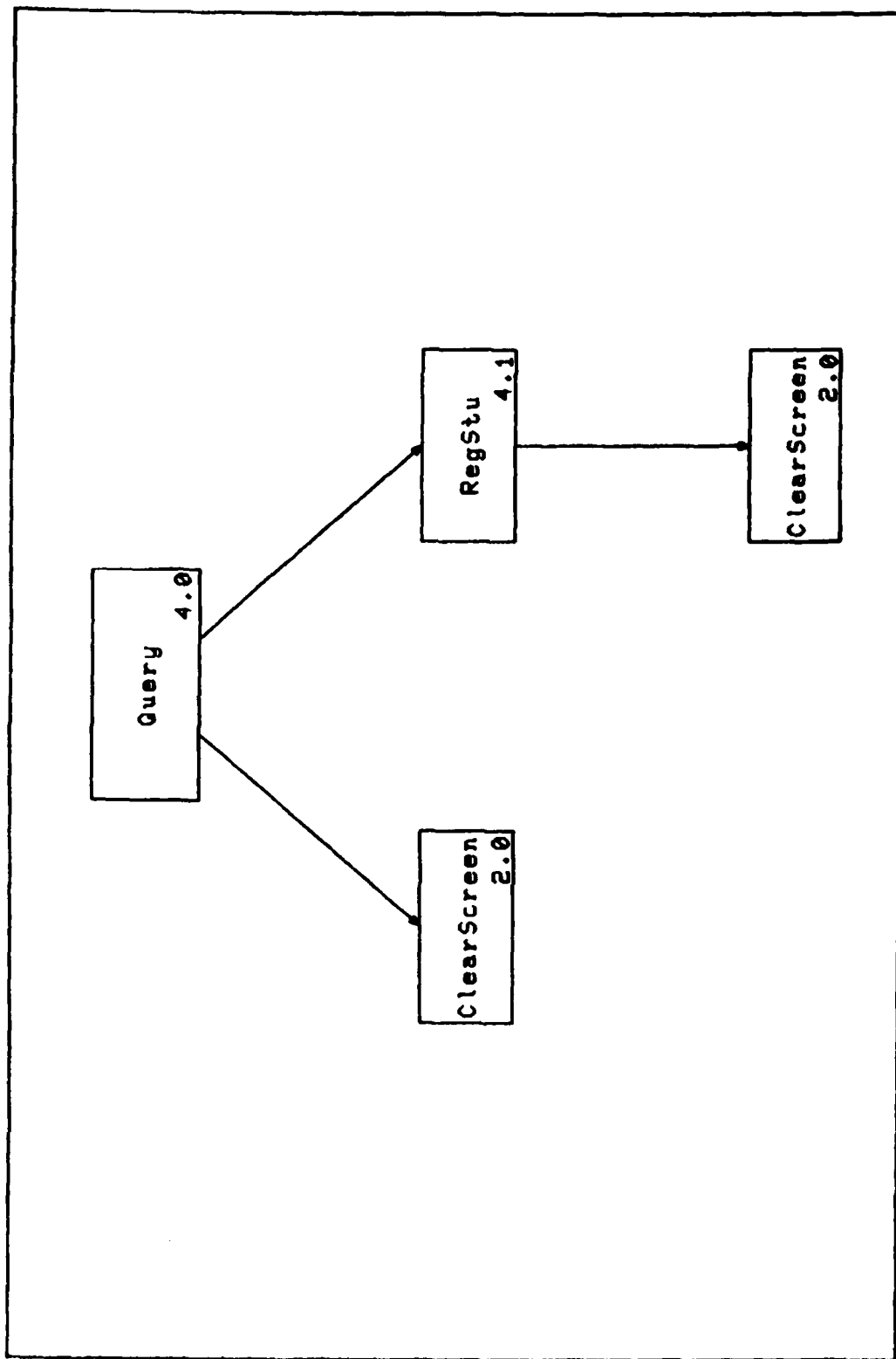


Figure 4.2 CAI - Query

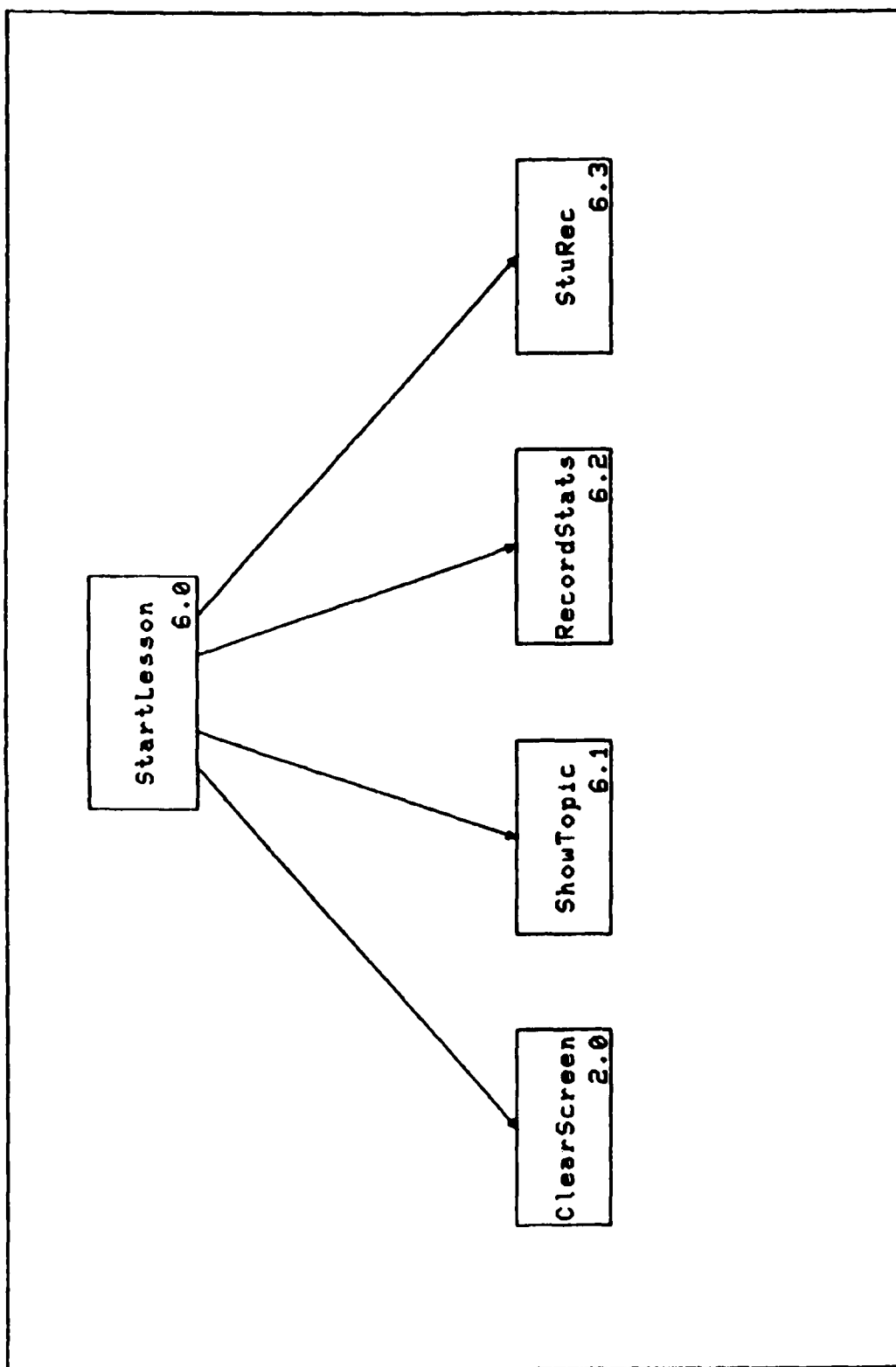


Figure 4.3 CAI - StartLesson

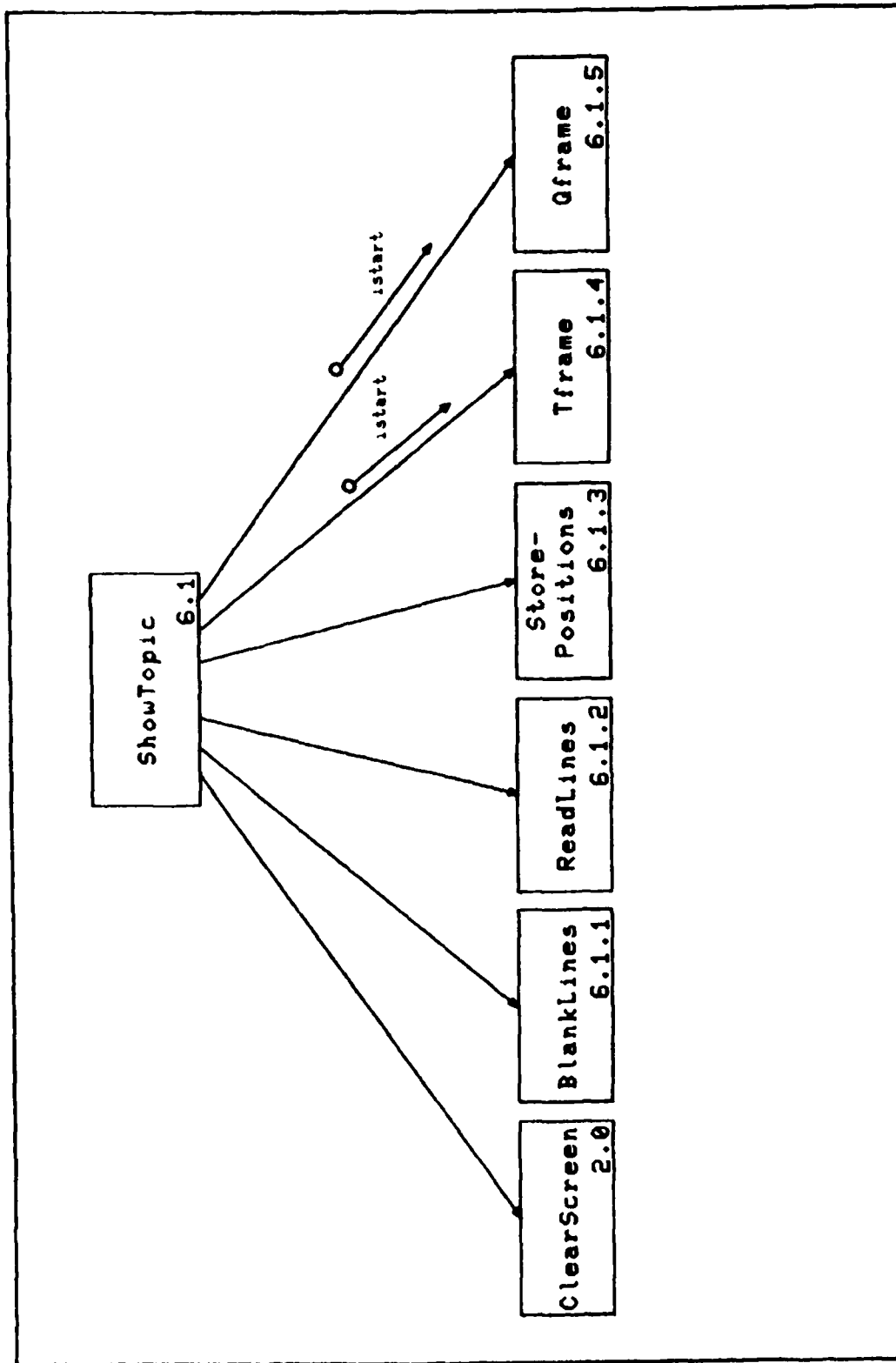


Figure 4.4 CAI - ShowTopic

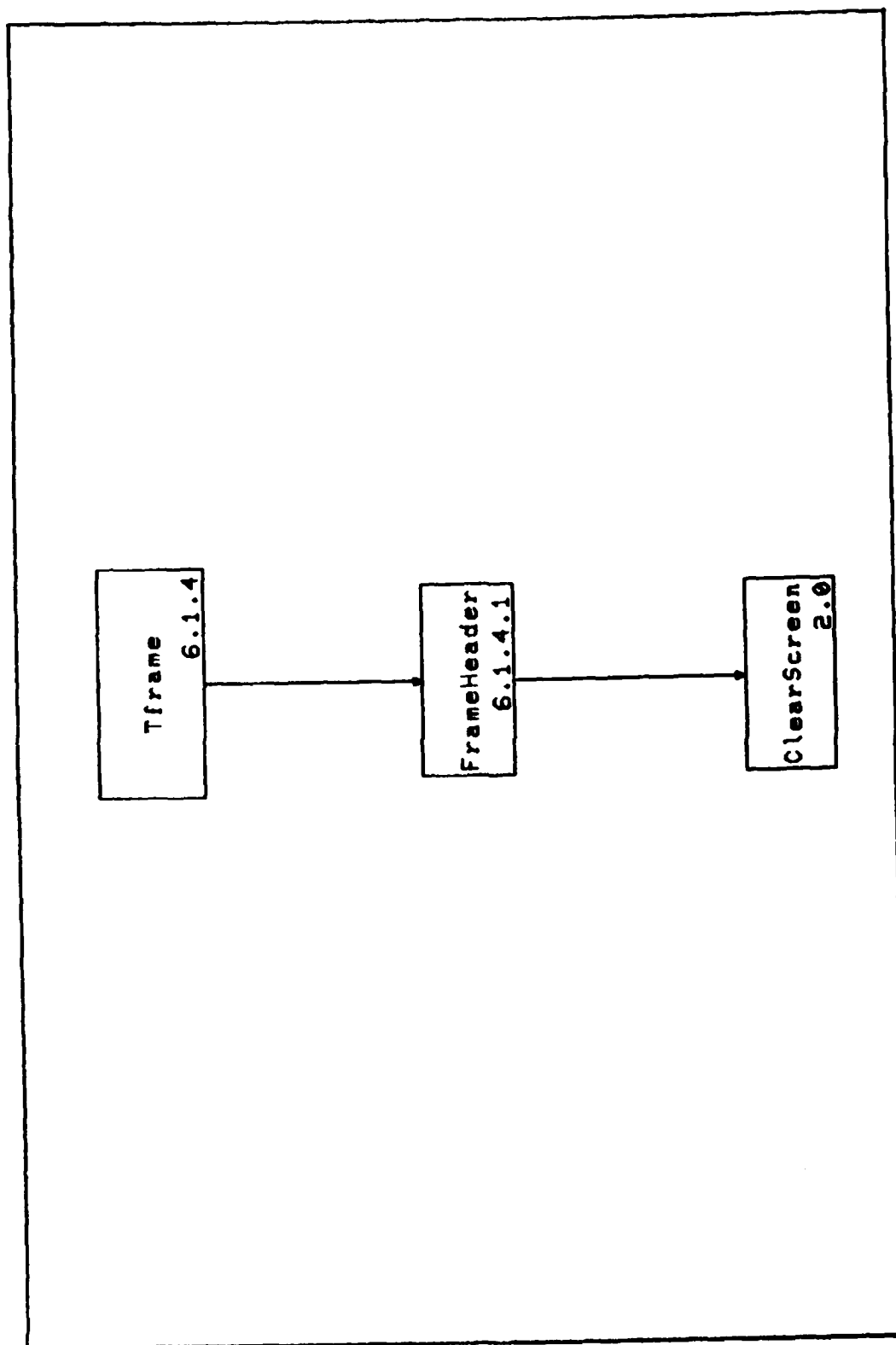


Figure 4.5 CAI - Tframe

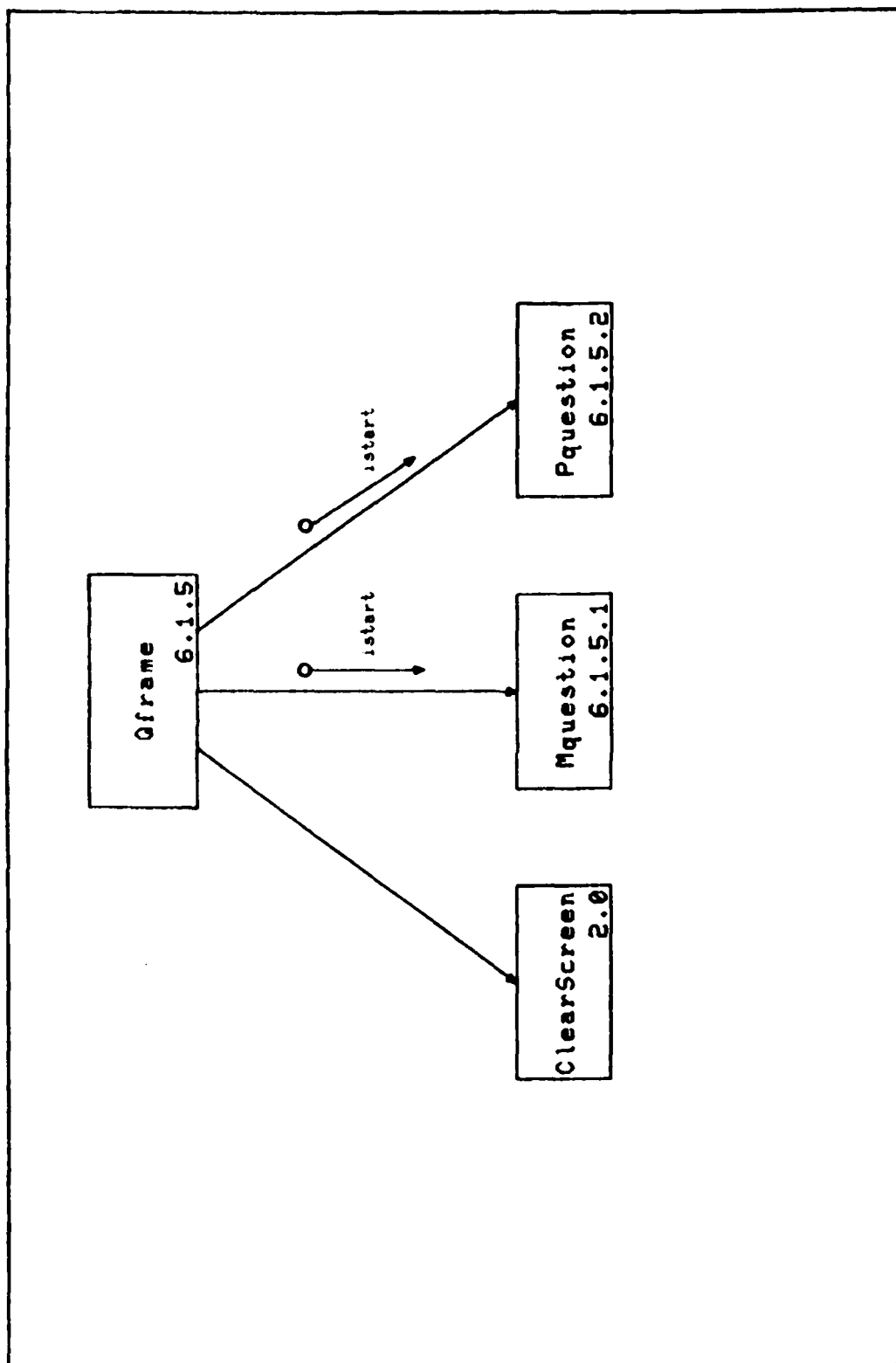


Figure 4.6 CAI - Qframe

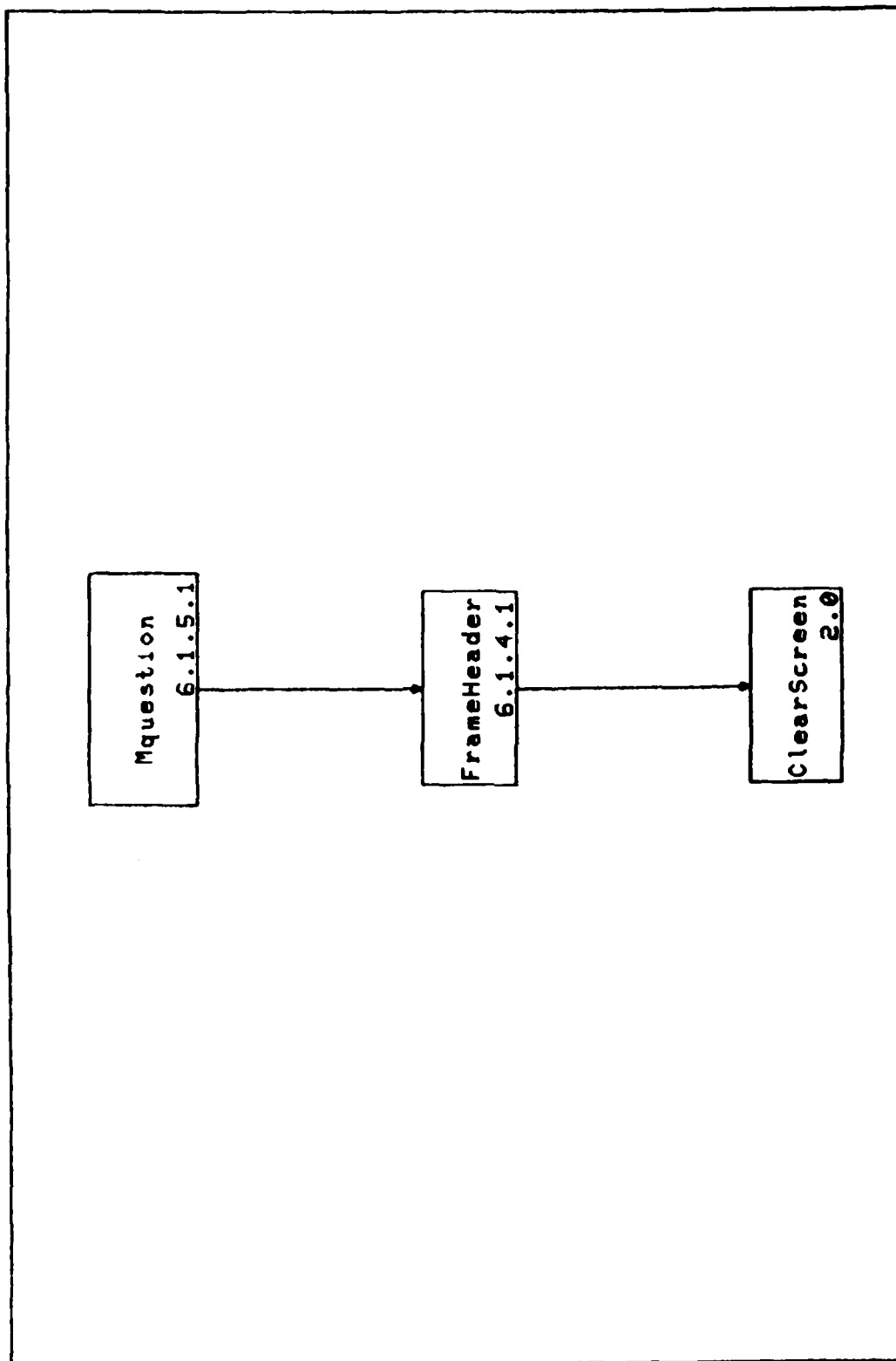


Figure 4.7 CAI - Mquestion

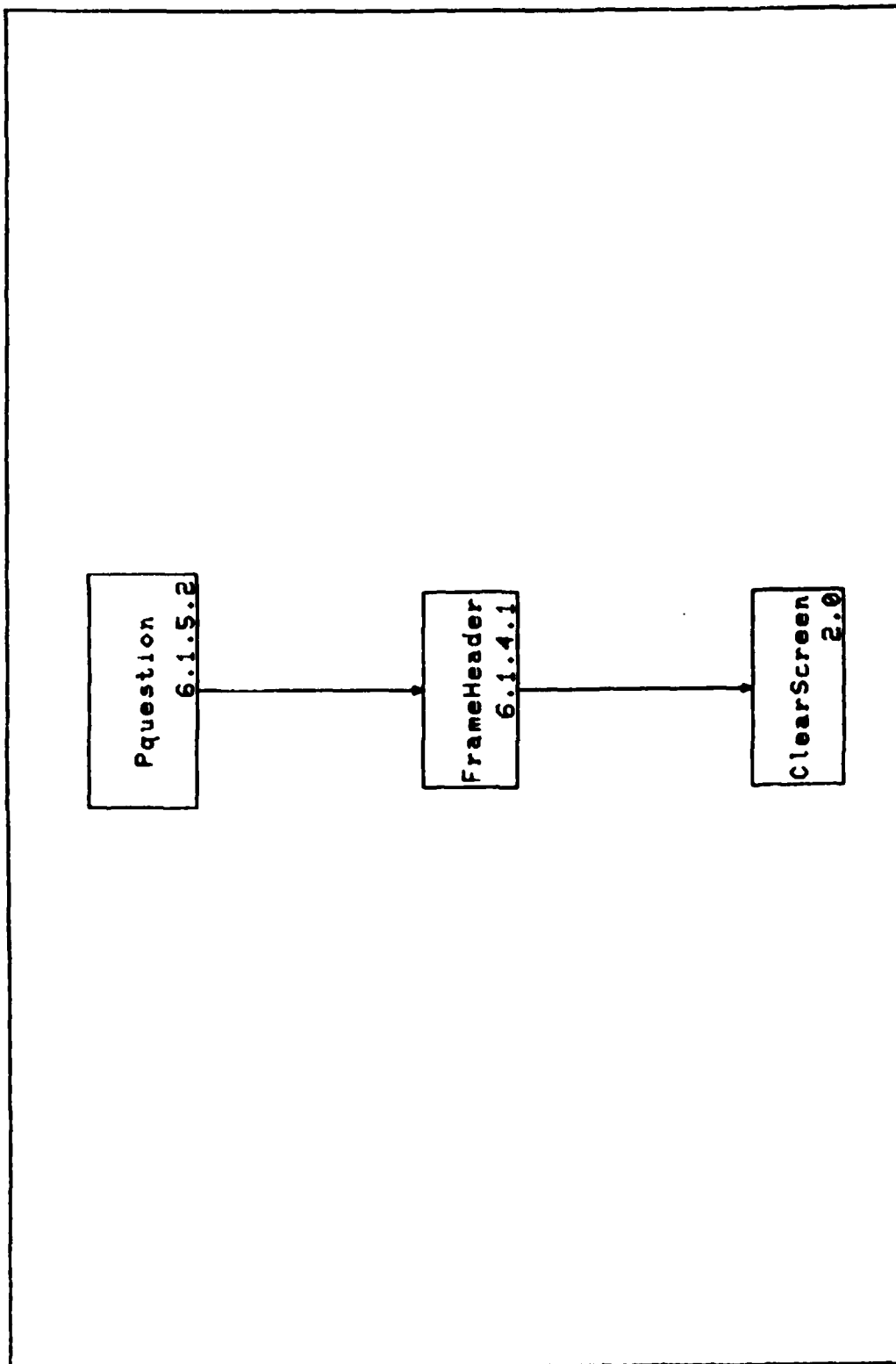


Figure 4.8 CAI - Pquestion

Status Program

Program Student_Status is the program that is used to present the current student status for all students recorded in file "STUDENT". The program is designed to accept and present any number of student records. This provides for the merging of several student files prior to running the program. There are two output formats for this program, "screen" and "hardcopy". The following is a breakdown and brief description of the program.

Structure Chart. The program is broken into a main program and five procedures, all of which are written in the Pascal programming language. A structure chart of this program is presented in Figure 4.9 of this chapter.

Flow Description. The flow of this program follows a straightforward path. The program begins by asking the user for the preferred method of report format, choices are either "screen" or "hardcopy". A header is then displayed and is followed by the student progress information. The structure of the report is in the format of "student identification number" followed by the word "passed" for every lesson that the student has successfully completed.

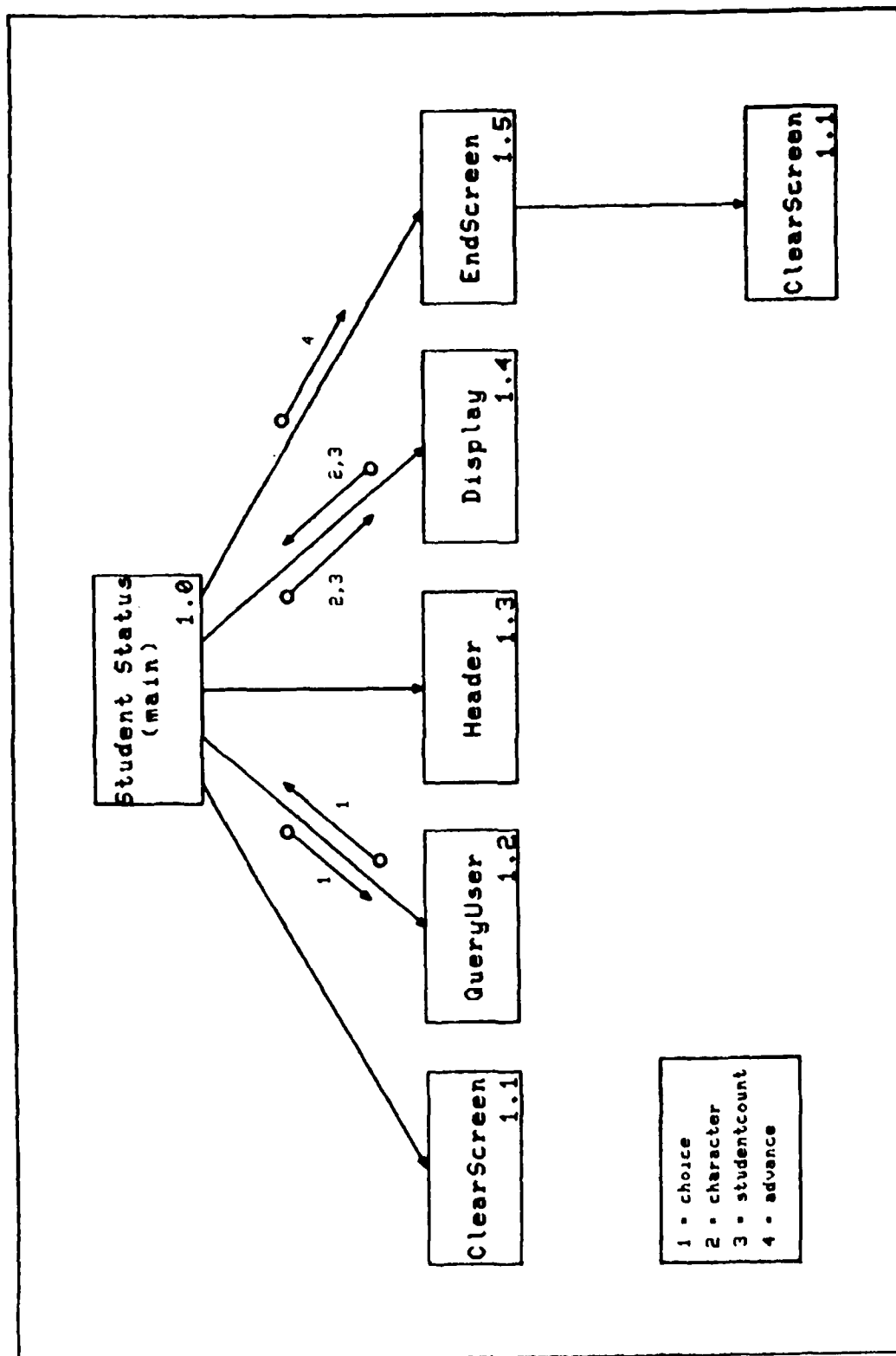


Figure 4.9 Student Status - Main

Statistics Program

Program CAI_Statistics is the program that is used to present the statistics collected on all questions asked during all course presentation sessions. The purpose of the program is to provide a means for the office of primary responsibility (OPR) at Keesler AFB to verify course content and effectiveness. The program is designed to accept and present statistics on as many as 150 different question frames. This restriction can be overcome by changing one line of source code (a constant value), if it becomes necessary. Several "STATS" files can be combined (and should be) before running this program. There are two output formats for this program, "screen" and "hardcopy". The following is a breakdown and brief description of the program.

Structure Charts. The program is broken into a main program and eight procedures, all of which are written in the Pascal programming language. Structure charts of this program are presented in Figures 4.10 thru 4.12 of this chapter.

Flow Description. The flow of this program follows a straightforward path. The program begins by asking the user for the preferred method of report format, choices are either "screen" or "hardcopy". A header is then displayed and is followed by internal reading and sorting routines. The output report is displayed in columns, giving all the needed statistics to the user. Items such as percent right and percent wrong help to validate questions.

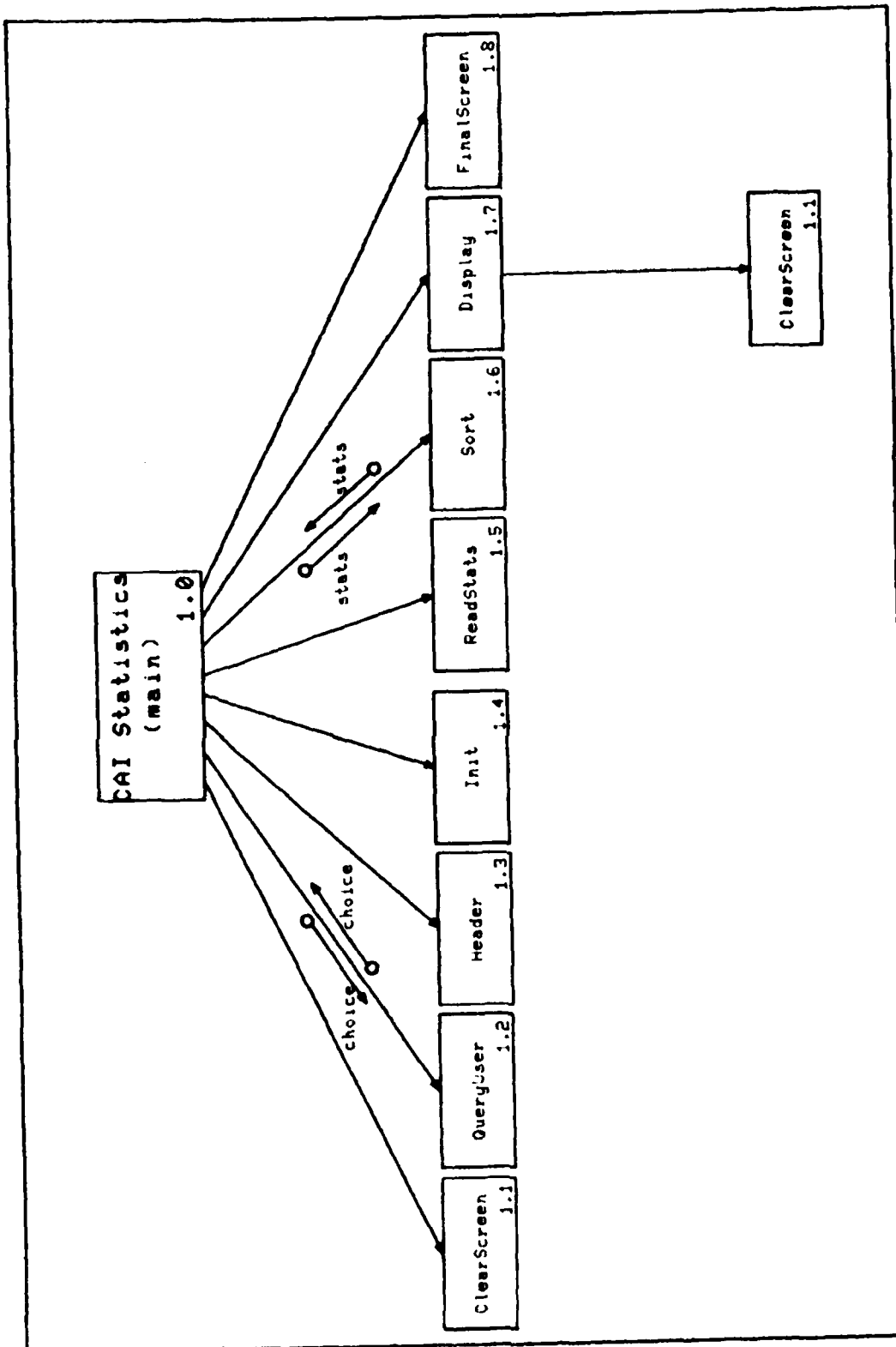


Figure 4.10 CAI Statistics - Main

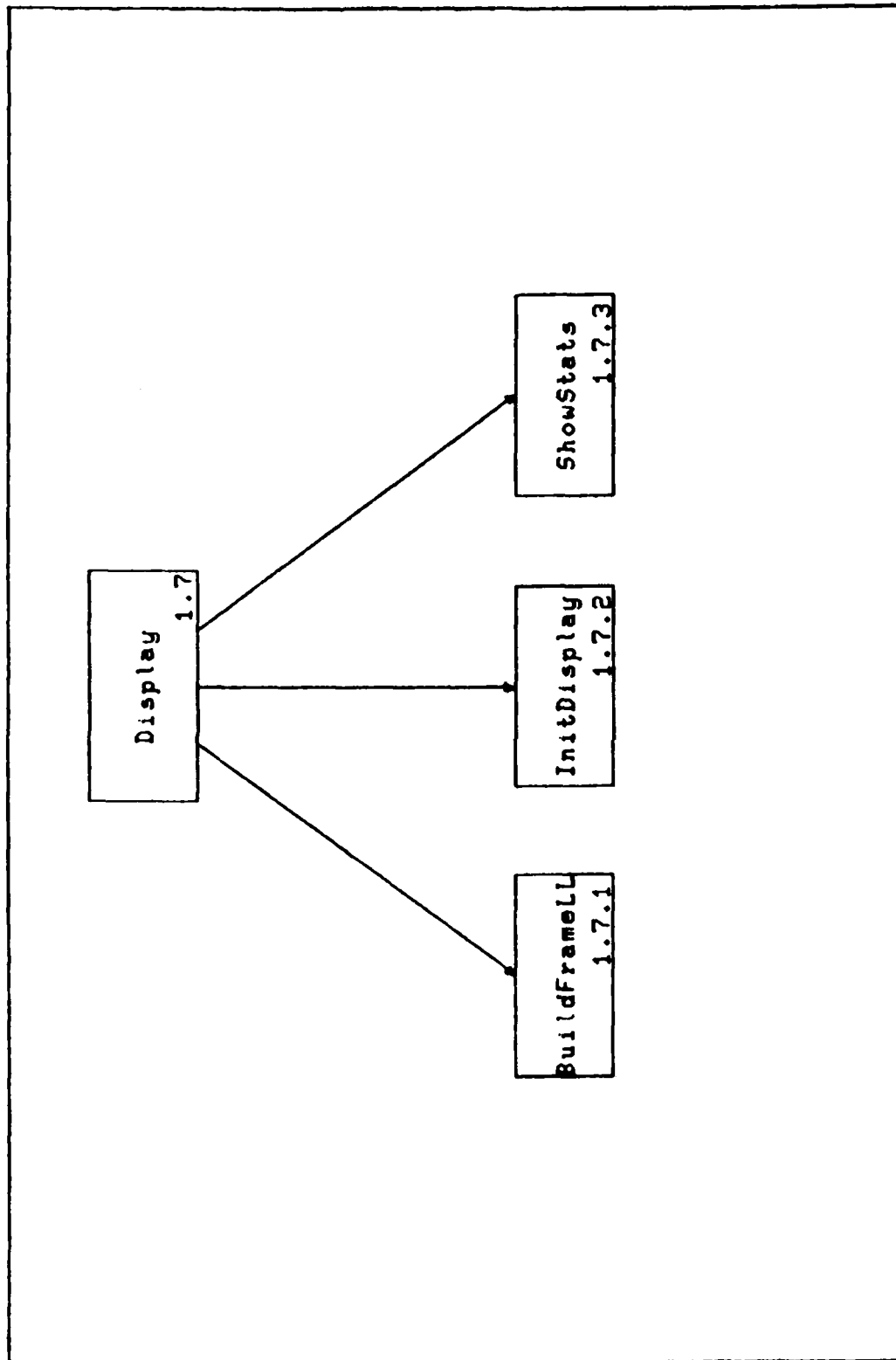


Figure 4.11 CAI Statistics - Display

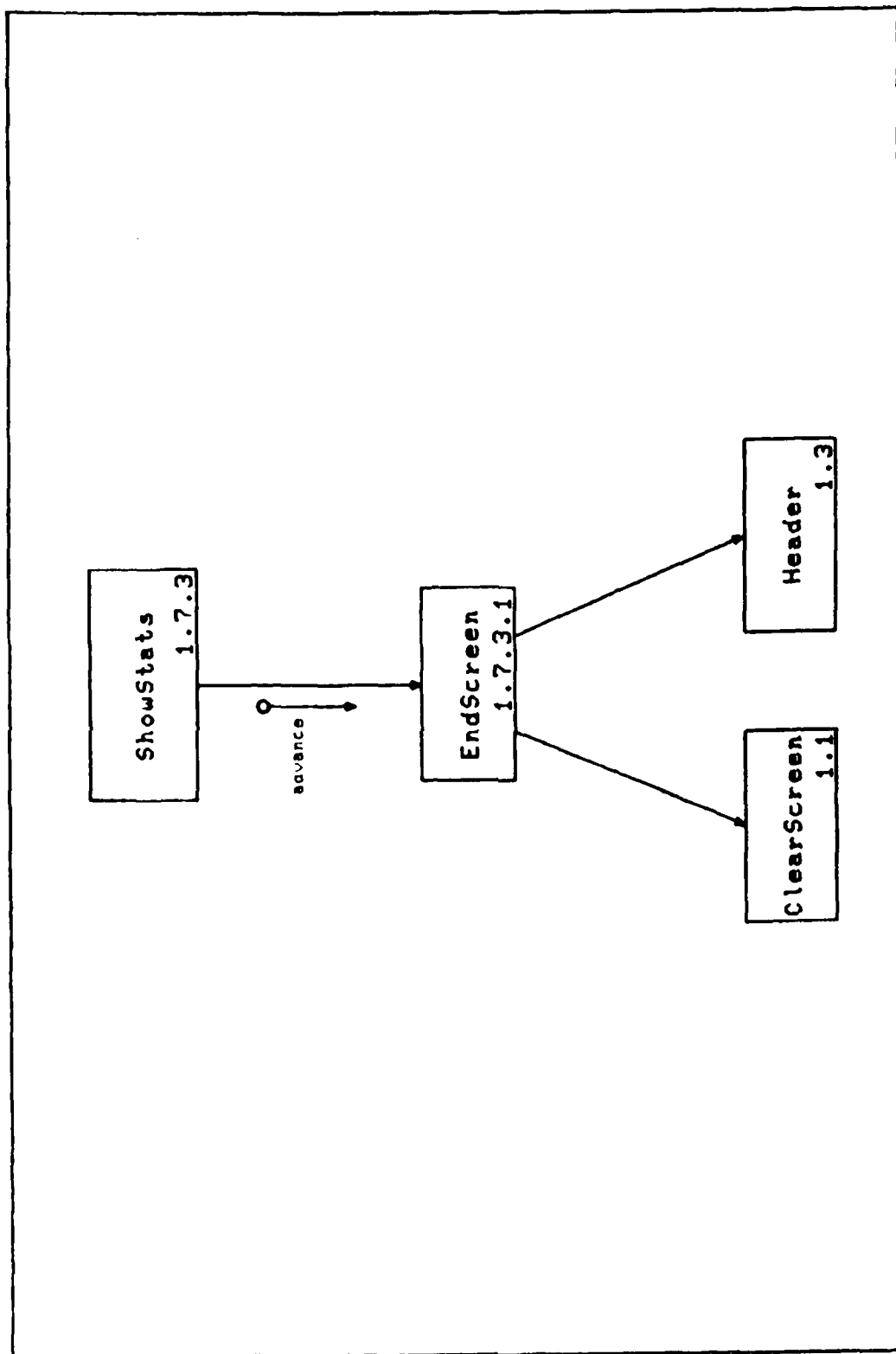


Figure 4.12 CAI Statistics - ShowStats

V. Conclusions and Recommendations

General Comments

The computer assisted instruction (CAI) package developed, tested, and implemented in this thesis effort presents an introduction to the "C" programming language. Although it does not get deep into fancy "C" language usage, it does serve its primary purpose of providing a strong base from which the student can build his/her "C" programming expertise. With a little initiative, the student will soon have the full power of the language at their disposal.

As was mentioned in chapter one, the primary goal of this study was to develop a course on the "C" programming language to be presented on the Zenith Z-100 microcomputer system. In order to achieve the stated goal a secondary goal had to be met, that of developing a software presentation system for the developed course material. This secondary goal provides the possibility of producing other courses for presentation on the Z-100 system.

Suggestions for Further Study

The existing presentation program is a good one as it stands, but certain enhancements would make it better. One such enhancement would be to add logic to allow for the asking of "fill in the blank" type questions. Another would be to allow the student to backup to a previously seen frame. One improvement in program control would be to read in the first frame of a topic, display it, and then read in

the rest of the topic while the student is reading the first frame. Currently, the student must wait nearly one minute before any topic material is displayed after they have chosen the topic from the topic menu.

Finally, the overall "C" course can be improved in several ways. Two of these are: provide for more branching to supplemental material and cover more of the capabilities of the "C" programming language. The course material and the programs used in conjunction with its use can be an effective means of getting introduced to the wonders of "C" programming.

Appendix A

Users Guide

Using Program "CAI"

Program CAI is the main program of this computer assisted instruction (CAI) package. The executable program is stored on "Disk 1" under the filename CAI.EXE. To start this program running, you need to boot the Zenith Z-100 microcomputer using the MS-DOS operating system. Remove the operating system disk from drive A, place "Disk 1" in drive A and "Disk 2" in drive B. Disk 2 contains the six lesson files of the C CAI course.

Once the disks are in place, type CAI in response to the A> prompt. The main CAI program will begin to execute and will prompt you for any further needed responses. One important item that deserves special mention is the student identification number that you will be prompted for during initial startup. This number is used to keep track of an individual's progress through the course. In order for it to be an effective feature of the package, the same sequence of characters must be entered each time you enter the CAI program.

Using Program "STUDENT STATUS"

Program Student_Status is designed for system training monitors. It is not for use by the students taking the course. This program will produce a report giving the current student status for each student recorded in file

"STUDENT" on "Disk 1". The executable program is stored on "Disk 1" under the filename STATUS.EXE. To start this program running, you need to boot the Zenith Z-100 microcomputer using the MS-DOS operating system. Replace the operating system disk in drive A with "Disk 1" of the CAI package. Once the disk is in place, type STATUS in response to the A> prompt. The Student_Status program will begin to execute and will prompt you for any further needed responses.

Using Program "CAI STATISTICS"

Program CAI_Statistics is designed for the office of primary responsibility (OPR) at Keesler AFB. It is not for use by the students taking the course. This program will produce a report giving statistics on all the C CAI course questions recorded in file "STATS" on "Disk 1". The executable program is stored on "Disk 1" under the filename VALIDATE.EXE. To start this program running, you need to boot the Zenith Z-100 microcomputer using the MS-DOS operating system. Replace the operating system disk in drive A with "Disk 1" of the CAI package. Once the disk is in place, type VALIDATE in response to the A> prompt. The CAI_Statistics program will begin to execute and will prompt you for any further needed responses.

Appendix B

Program Listings

Program "CAI"

(**** THIS PROGRAM WAS WRITTEN IN PARTIAL FULFILLMENT OF A MASTERS THESIS *****)

(*****)

```
*   Date: 8/1/85                                           *
*   Version: 1.0                                           *
*                                                         *
*   Title: Program CAI                                     *
*   Filename: CAI.PAS                                       *
*   Coordinator: Capt Frank W. DeMarco                     *
*   Project: Masters Thesis                                 *
*   Operating System: MS-DOS                               *
*   Language: Pascal                                       *
*   Use: Compile and link with PASCAL.LIB using MS-Pascal  *
*   compiler and linker. *
*   Contents: Program CAI - Main Driver.                   *
*       Procedure ClearScreen - Clears 2-100 terminal      *
*       screen. *
*       Procedure RegStu - Registers a first time student. *
*       Procedure Query - Reads in "STUDENT" file, prompts *
*       student for identification number, and checks the  *
*       ID number against current student list. *
*       Procedure StartEnd - Reads and displays files      *
*       "INTRO" at start of program and "EXIT" at end of  *
*       program. *
*       Procedure Select - Reads and displays file "MENU", *
*       prompts the student for choice of lesson to be    *
*       shown. *
*       Procedure ShowTopic - Driver of procedures that    *
*       display topic material. *
*       Procedure BlankLines - Initializes area where      *
*       topic material is stored to blanks. *
*       Procedure Readlines - Reads in topic that the      *
*       student chose to view. *
*       Procedure StorePositions - Builds an array of line  *
*       positions where frames begin within the topic. *
*       Procedure FrameHeader - Displays a frame header    *
*       for a frame. *
*       Procedure Tframe - Displays a text type frame. *
*       Procedure Qframe - Driver for the procedures that  *
*       display and handle question type frames. *
*       Procedure Mquestion - Displays and handles multiple *
*       choice type question frames. *
*       Procedure Pquestion - Displays and handles pick    *
*       type question frames (true/false and yes/no). *
*       Procedure RecordStats - Reads file "STATS" and     *
*       adds statistical data from current session. *
*       Procedure StuRec - Writes updated student course   *
*       progress data to file "STUDENT". *
*       Procedure StartLesson - Displays topic choices    *
*       for a lesson, prompts student for choice of topic *
*       to be shown. Driver of procedures that display    *
*       lesson material and update *
*                                                         *
```

```

*                               statistical & student progress files.      *
*                               *                                           *
*   Function: The purpose of this program is to present material on the "C" *
*             programming language. It is intended for use by the 3300 Tech- *
*             nical Training Wing in support of its mission. The office of  *
*             primary responsibility for this course is the CAI Plans Branch *
*             (3300 TCHTW/TTGXZ) at Keesler AFB, MS 39534                  *
*                               *                                           *
*****

```

```

{*****
*   Date: 8/1/85                                                         *
*   Version: 1.0                                                         *
*                               *                                           *
*   Name: program CAI                                                    *
*   Module number: 1.0                                                  *
*   Description: Main driver of program                                  *
*   Passed Variables: None                                              *
*   Returns: None                                                       *
*   Global Variables Used: studentcount, choice                        *
*   Global Variables Changed: None                                       *
*   Files Read: None                                                    *
*   Files Written: None                                                  *
*   Modules Called: ClearScreen, StartEnd, Query, Select, StartLesson *
*   Calling modules: None                                               *
*                               *                                           *
*   Author: Capt Frank W. DeMarco                                       *
*   History:                                                             *
*     1.0 Frank W. DeMarco 8/1/85 - input original code                *
*****}

```

program CAI (input,output);

```

const
  MAXSTUDENTS = 20;
  MAXLESSONS = 6;
  MAXTOPICS = 5;
  VLESSON = '6';
  VTOPIC = '5';
  ALINEP1 = '*****';
  ALINEP2 = '*****';
  ABLANKS = '*';
  BLANKSA = '          *';

type
  tofile = TEXT;
  roll = record
    studentnumber : packed array [1..11] of char;
    studentname : packed array [1..29] of char;
    lessons : packed array [1..MAXLESSONS] of char;
    topics : array [1..MAXLESSONS, 1..MAXTOPICS] of char;
  end;
  roster = array [1..MAXSTUDENTS] of roll;

```

```

lstat = array [1..MAXLESSONS] of char;
displayln = packed array [1..80] of char;
lessonlines = array [1..500, 1..80] of char;
menulines = array [1..22, 1..80] of char;
tstat = array [1..MAXTOPICS] of char;

```

```

var
  iomessage, student, statfile, menu, lesson, temp1, temp2 : iofile;
  advance, linecount, studentcount : integer;
  choice, lchoice : char;
  npupil : roll;
  rpupil : roster;
  lessonstat : lstat;
  println : displayln;
  lessonln : lessonlines;
  menuln : menulines;
  topicstat : tstat;

```

```

{*****}
*   Date: 8/1/85                                           *
*   Version: 1.0                                           *
*                                                         *
*   Name: procedure ClearScreen                           *
*   Module number: 2.0                                     *
*   Description: Clears Z-100 terminal screen and sets "no-wrap" on EOL. *
*   Passed Variables: None                                *
*   Returns: None                                          *
*   Global Variables Used: None                            *
*   Global Variables Changed: None                        *
*   Files Read: None                                       *
*   Files Written: None                                    *
*   Modules Called: None                                   *
*   Calling modules: program CA1, StartEnd, Query, Select, RegStu, *
*                   StartLesson, ShowTopic, Oframe, FrameHeader *
*                                                         *
*   Author: Capt Frank W. DeMarco                         *
*   History:                                               *
*       1.0 Frank W. DeMarco 8/1/85 - input original code *
*****}

```

```

procedure ClearScreen;

```

```

begin  (* Procedure ClearScreen *)

```

```

    write (chr(27),'H',chr(27),'J',chr(27),'W')

```

```

end;  (* Procedure ClearScreen *)

```

```

{*****}
*   Date: 8/1/85                                           *
*   Version: 1.0                                           *
*                                                         *

```

```

*   Name: procedure RegStu
*   Module number: 4.1
*   Description: Registers a first time student.
*   Passed Variables: None
*   Returns: None
*   Global Variables Used: npupil, studentcount
*   Global Variables Changed: npupil, studentcount
*   Files Read: None
*   Files Written: None
*   Modules Called: ClearScreen
*   Calling modules: Query
*
*   Author: Capt Frank W. DeMarco
*   History:
*   1.0 Frank W. DeMarco 8/1/85 - input original code
*****

```

```

procedure RegStu;

```

```

var

```

```

    i, j : integer;

```

```

begin (* Procedure RegStu *)

```

```

    for j := 1 to 11 do
        npupil.studentnumber[j] := ' ';

```

```

    for j := 1 to 28 do
        npupil.studentname[j] := ' ';

```

```

    for j := 1 to MAXLESSONS do
        npupil.lessons[j] := ' ';

```

```

    for i := 1 to MAXLESSONS do
        begin

```

```

            for j := 1 to MAXTOPICS do
                npupil.topics[i,j] := ' ';

```

```

            end;

```

```

        writeln;

```

```

        writeln('Since this is your first time into this course, I have a few ');

```

```

        writeln('administrative matters to take care of. ');

```

```

        writeln;

```

```

        writeln('Please enter your first name: ');

```

```

        write(' (Max. of 10 characters) >>>>> ');

```

```

        i := 1;

```

```

        while not (eoln) and (i < 11) do

```

```

            begin

```

```

                read (npupil.studentname[i]);

```

```

                i := i + 1;

```

```

            end;

```

```

        if (eoln) and (i < 11) then

```

```

            begin

```

```

                for i := i to 10 do

```

```

                    npupil.studentname[i] := '*';

```

```

        readln
    end
else
    readln;

writeln;
writeln;

writeln('Please enter your middle initial: ');
write(' (Max. of 1 character) >>>>>>>>> ');

i := 11;
while not (eoln) and (i < 12) do
begin
    read (npupil.studentname[i]);
    i := i + 1
end;
if (npupil.studentname[i] in ['a'..'z', 'A'..'Z']) then
begin
    npupil.studentname[i] := '.';
    readln
end
else
begin
    npupil.studentname[i] := '*';
    npupil.studentname[i] := '*';
    readln
end;

writeln;
writeln;

writeln('Please enter your last name: ');
write(' (Max. of 16 characters) >>>> ');

i := 13;
while not (eoln) and (i < 29) do
begin
    read (npupil.studentname[i]);
    i := i + 1
end;
if (eoln) and (i < 29) then
begin
    for i := i to 28 do
        npupil.studentname[i] := '*';
    readln
end
else
    readln;

writeln;
writeln;

```

```
writeln('Now for the most important part. ');
writeln('Please enter your unique, personal student identification number: ');
write(' (Max. of 11 characters) >>>>> ');
```

```
i := 1;
while not (eoln) and (i < 12) do
begin
  read (npupil.studentnumber[i]);
  i := i + 1;
end;
if (eoln) and (i < 12) then
begin
  for i := i to 11 do
    npupil.studentnumber[i] := '*';
  readln
end
else
  readln;
```

```
for i := 1 to MAXLESSONS do
begin
  npupil.lessons[i] := '-';
  for j := 1 to MAXTOPICS do
    npupil.topics[i,j] := '-';
end;
```

```
studentcount := studentcount + 1
```

```
end; (* Procedure RegStu *)
```

```
(*****
*   Date: 8/1/85
*   Version: 1.0
*
*   Name: procedure Query
*   Module number: 4.0
*   Description: Reads in "STUDENT" file, prompts student for student identi-
*                 fication number, and checks the ID number against current
*                 student list.
*   Passed Variables: None
*   Returns: None
*   Global Variables Used: rpupil, studentcount, npupil
*   Global Variables Changed: rpupil, studentcount, npupil
*   Files Read: student
*   Files Written: None
*   Modules Called: Clear Screen, RegStu
*   Calling modules: program CAI
*
*   Author: Capt Frank W. DeMarco
*   History:
*   1.0 Frank W. DeMarco 8/1/85 - input original code
*****)
```

```

procedure Query;

var
  i, ii, j : integer;

  qfound : boolean;
  character : char;

begin (* Procedure Query *)

  assign (student,'student');
  reset (student);
  character := ' ';
  if not (eof(student)) then
    read (student,character);

  for i := 1 to MAXSTUDENTS do
    begin
      for j := 1 to 11 do
        rpupil[i].studentnumber[j] := ' ';
      for j := 1 to 28 do
        rpupil[i].studentname[j] := ' ';
      for j := 1 to MAXLESSONS do
        rpupil[i].lessons[j] := ' ';
      for ii := 1 to MAXLESSONS do
        begin
          for j := 1 to MAXTOPICS do
            rpupil[i].topics[ii,j] := ' ';
          end;
        end;
    end;

  i := 1;
  studentcount := 0;
  while (character = '>') and not (eof(student)) do
    begin
      studentcount := studentcount + 1;
      while not (eoln(student)) do
        begin
          for j := 1 to 11 do
            read (student,rpupil[i].studentnumber[j]);
          for j := 1 to 28 do
            read (student,rpupil[i].studentname[j]);
          for j := 1 to MAXLESSONS do
            read (student,rpupil[i].lessons[j]);
          for ii := 1 to MAXLESSONS do
            begin
              for j := 1 to MAXTOPICS do
                read (student,rpupil[i].topics[ii,j]);
              end;
            end;
          i := i + 1;
        end;
      if not (eof(student)) then
        readln (student);
    end;
  end;

```



```

        if not (eof(student)) then
            read (student,character);
        end;

ClearScreen;

write('Please enter your student identification number: ');
write('(Max. of 11 characters) >>>>> ');

i := 1;
while not (eoln) and (i < 12) do
    begin
        read (npupil.studentnumber[i]);
        i := i + 1
    end;
if (eoln) and (i < 12) then
    begin
        for i := i to 11 do
            npupil.studentnumber[i] := '*';
        end;
    readln;

    i := 1;
    qfound := false;
    while (i < 21) do
        begin
            if (npupil.studentnumber = rpupil[i].studentnumber) then
                begin
                    qfound := true;
                    npupil.studentname := rpupil[i].studentname;
                    npupil.lessons := rpupil[i].lessons;
                    for ii := 1 to MAXLESSONS do
                        begin
                            for j := 1 to MAXTOPICS do
                                npupil.topics[ii,j] := rpupil[i].topics[ii,j]
                            end;
                        end;
                    i := i + 1
                end;
            if not (qfound) and (studentcount < MAXSTUDENTS) then
                begin
                    ClearScreen;
                    writeln ('NO MATCH FOUND');
                    RegStu;
                    ClearScreen
                end
            else
                if not (qfound) and (studentcount >= MAXSTUDENTS) then
                    begin
                        ClearScreen;
                        studentcount := studentcount + 1;
                        writeln('Sorry, but my class roster shows a "Full" class.');
```

```

        writeln('Please see your training monitor for a new student disk.');
```

writeln;

```
        writeln('END OF PROGRAM')
    end;
```

close (student)

```
end; (* Procedure Query *)
```

```

(*****
*   Date: 8/1/85
*   Version: 1.0
*
*   Name: procedure StartEnd
*   Module number: 3.0
*   Description: Reads and displays files "INTRO" at start of program and
*               "EXIT" at end of program.
*   Passed Variables: code
*   Returns: None
*   Global Variables Used: println, linecount, advance
*   Global Variables Changed: println, linecount, advance
*   Files Read: intro, exit
*   Files Written: None
*   Modules Called: ClearScreen
*   Calling modules: program CAI
*
*   Author: Capt Frank W. DeMarco
*   History:
*       1.0 Frank W. DeMarco 8/1/85 - input original code
*****)
```

```

procedure StartEnd(code : char);

var
    character : char;
    i : integer;

begin (* Procedure StartEnd *)

    case code of
        'S' : assign (iomessage,'intro');
        'E' : assign (iomessage,'exit')
    end;

    reset (iomessage);
    read (iomessage,character);
    linecount := 0;

    repeat

        while (character = '#') and not (eof(iomessage)) do
            begin
                for i := 1 to 80 do

```

```

        println[i] := ' ';
        readln (iomessage,println);
        linecount := linecount + 1;
        writeln (println);
        if not (eof(iomessage)) then
            read (iomessage,character)
        end;

if (character = '!') then
begin
    advance := 23 - linecount;
    linecount := 0;
    for i := 1 to advance do
        writeln;
    for i := 1 to 27 do
        write (' ');
    write ('Press RETURN to continue. ');
    readln;
    if not (eof(iomessage)) then
        begin
            readln (iomessage);
            if not (eof(iomessage)) then
                read (iomessage,character)
            end;
        end;
    ClearScreen
end;
until (eof(iomessage));

close (iomessage)

end; (* Procedure StartEnd *)

(*****
*   Date: 9/1/85
*   Version: 1.0
*
*   Name: procedure Select
*   Module number: 5.0
*   Description: Reads and displays file "MENU", prompts the student for
*               choice of lesson to be shown.
*   Passed Variables: None
*   Returns: None
*   Global Variables Used: lessonstat, println, choice
*   Global Variables Changed: lessonstat, println, choice
*   Files Read: menu
*   Files Written: None
*   Modules Called: ClearScreen
*   Calling modules: program CAI
*
*   Author: Capt Frank W. DeMarco
*   History:
*   1.0 Frank W. DeMarco 9/1/85 - input original code
*****)

```

```

procedure Select;

var
  character : char;
  i, j : integer;

begin (* Procedure Select *)

  assign (menu,'menu');

  for i := 1 to MAXLESSONS do
    lessonstat[i] := npupil.lessons[i];

  repeat

  reset (menu);
  read (menu,character);

  j := 0;
  while (character in ['*', '@']) and not (eof(menu)) do
    begin
      readln (menu,println);

      if (character = '*') then
        begin
          write ('*');
          for i := 2 to 78 do
            write (println[i]);
          writeln (println[79])
        end
      else
        begin
          j := j + 1;
          write ('*');
          for i := 2 to 8 do
            write (println[i]);
          write (lessonstat[j]);
          for i := 10 to 78 do
            write (println[i]);
          writeln (println[79])
        end;

      if not (eof(menu)) then
        read (menu,character)
      end;

  writeln;
  write ('ENTER THE NUMBER OF YOUR CHOICE OR "X" TO EXIT THE CAI PROGRAM: ');
  readln (choice);
  if (choice in ['1'..VLESSON, 'x', 'X']) then
    ClearScreen
  else

```

```

begin
  ClearScreen;
  writeln ('Sorry, ',choice,' is not a valid response. Please try again.')
end;

until (choice in ['1'..VLESSON,'X','X']);

if (choice in ['1'..VLESSON]) then
  writeln ('You have chosen lesson number ',choice,'. Thank you.')
else
  writeln ('OK, I will now return you to the operating system.');
```

close (menu)

```

end;  (* Procedure Select *)

(*****
*   Date: 8/1/85
*   Version: 1.0
*
*   Name: procedure ShowTopic
*   Module number: 6.1
*   Description: Driver of procedures that display topic material.
*   Passed Variables: None
*   Returns: None
*   Global Variables Used: lessonln, npupil, topicstat
*   Global Variables Changed: npupil, topicstat
*   Files Read: None
*   Files Written: None
*   Modules Called: ClearScreen, BlankLines, ReadLines, StorePositions,
*                   Tframe, Qframe
*   Calling modules: StartLesson
*
*   Author: Capt Frank W. DeMarco
*   History:
*   1.0 Frank W. DeMarco 8/1/85 - input original code
*****)

procedure ShowTopic;

const
  MINSORE = 70.0;

type
  position = record
    framenum : integer;
    ivalue : integer;
  end;
  topictitle = packed array [1..30] of char;

var
  lplace : array [1..50] of position;
  tname : topictitle;
```

```

i, j, k, istart, nextframe, frame : integer;
nummasked, numright, score : real;
sfound, test : boolean;
ftype : char;

```

```

(*****
*   Date: 8/1/85
*   Version: 1.0
*
*   Name: procedure BlankLines
*   Module number: 6.1.1
*   Description: Initializes area where topic material is stored to blanks.
*   Passed Variables: None
*   Returns: None
*   Global Variables Used: lessonln, tname, lplace
*   Global Variables Changed: lessonln, tname, lplace
*   Files Read: None
*   Files Written: None
*   Modules Called: None
*   Calling modules: ShowTopic
*
*   Author: Capt Frank W. DeMarco
*   History:
*   1.0 Frank W. DeMarco 8/1/85 - input original code
*****)

```

```

procedure BlankLines:

```

```

var indexi, indexj : integer;

```

```

begin (* Procedure BlankLines *)

```

```

  for indexi := 1 to 500 do
    begin
      for indexj := 1 to 80 do
        lessonln[indexi,indexj] := ' ';
      end;
    end;

```

```

  for indexi := 1 to 30 do
    tname[indexi] := ' ';

```

```

  for indexi := 1 to 50 do
    begin
      lplace[indexi].framenum := 0;
      lplace[indexi].lvalue := 0;
    end;

```

```

end; (* Procedure BlankLines *)

```

```

(*****
*   Date: 8/1/85
*   Version: 1.0
*

```

```

*   Name: procedure ReadLines
*   Module number: 6.1.2
*   Description: Reads in topic that the student chose to view.
*   Passed Variables: None
*   Returns: None
*   Global Variables Used: lessonIn, lchoice, tname
*   Global Variables Changed: lessonIn, tname
*   Files Read: lesson
*   Files Written: None
*   Modules Called: None
*   Calling modules: ShowTopic
*
*   Author: Capt Frank W. DeMarco
*   History:
*   1.0 Frank W. DeMarco 8/1/85 - input original code
*****

```

```

procedure ReadLines;

```

```

var

```

```

    rfound : boolean;
    jvalue , ivalue : integer;

```

```

begin (* Procedure ReadLines *)

```

```

    writeln;
    writeln ('One moment please...');

```

```

    rfound := false;
    reset (lesson);

```

```

    repeat
        read (lesson,lessonIn[1,1]);
        if (lessonIn[1,1] = lchoice) then
            rfound := true
        else
            readln(lesson);
    until (rfound);

```

```

    j := 2;
    while not (eoln(lesson)) do
        begin
            read (lesson,lessonIn[1,j]);
            j := j + 1;
        end;

```

```

    readln (lesson);
    i := 2;
    read (lesson,lessonIn[i,1]);

```

```

    j := 2;

```

```

while (lessonln[i,1] = lchoice) and not (eof(lesson)) do
begin
  while not (eoln(lesson)) do
  begin
    read (lesson,lessonln[i,j]);
    j := j + 1
  end;
  readln (lesson);
  i := i + 1;
  j := 1;
  if not (eof(lesson)) then
    read (lesson,lessonln[i,j]);
  j := j + 1
end;

ivalue := 1;
for jvalue := 16 to 45 do
begin
  tname[ivalue] := lessonln[1,jvalue];
  ivalue := ivalue + 1;
end;

end;  (* Procedure ReadLines *)

(*****
*   Date: 8/1/95
*   Version: 1.0
*
*   Name: procedure StorePositions
*   Module number: 6.1.3
*   Description: Builds an array of line positions where frames begin within
*               the topic.
*   Passed Variables: None
*   Returns: None
*   Global Variables Used: lessonln, i, j, k, lplace, lchoice
*   Global Variables Changed: i, j, k, lplace
*   Files Read: None
*   Files Written: None
*   Modules Called: None
*   Calling modules: ShowTopic
*
*   Author: Capt Frank W. DeMarco
*   History:
*   1.0 Frank W. DeMarco 8/1/95 - input original code
*****)

procedure StorePositions;

var
  *number, jval : integer;

begin  (* Procedure StorePositions *)

```



```

i := 1;
j := 2;
k := 0;

repeat
  if (lessonIn[i,j] = '1') then
    begin
      k := k + 1;
      fnumber := 0;
      for jval := 9 to 11 do
        fnumber := (10 * fnumber) + ((ord(lessonIn[i,jval])) - ord('0'));
      lplace[k].framenum := fnumber;
      lplace[k].lvalue := i;
      i := i + 1
    end
  else
    i := i + 1;
until (lessonIn[i,1] <> lchoice);

k := k + 1;          (* This marks the end *)
lplace[k].framenum := -1; (* of the array *)

end;  (* Procedure StorePositions *)

```

```

(*****
*   Date: 8/1/85
*   Version: 1.0
*
*   Name: procedure FrameHeader
*   Module number: 7.0
*   Description: Displays a frame header for a frame.
*   Passed Variables: None
*   Returns: None
*   Global Variables Used: None
*   Global Variables Changed: None
*   Files Read: None
*   Files Written: None
*   Modules Called: ClearScreen
*   Calling modules: Tframe, Mquestion, Pquestion
*
*   Author: Capt Frank W. DeMarco
*   History:
*   1.0 Frank W. DeMarco 8/1/85 - input original code
*****)

```

```

procedure FrameHeader;

begin  (* Procedure FrameHeader *)

  ClearScreen;

  writeln (ALINEP1,ALINEP2);

```

```
writeln ('* Lesson #',choice,' * Topic #',lchoice,' * Title: ',tname,
        ' * Frame: ',frame:3,' *');
writeln (ALINEP1,ALINEP2)
```

```
end; (* Procedure FrameHeader *)
```

```
(*****
*   Date: 8/1/85
*   Version: 1.0
*
*   Name: procedure Tframe
*   Module number: 6.1.4
*   Description: Displays a text type frame.
*   Passed Variables: istart
*   Returns: None
*   Global Variables Used: advance, linecount, lessonln, nextframe
*   Global Variables Changed: advance, linecount, nextframe
*   Files Read: None
*   Files Written: None
*   Modules Called: FrameHeader
*   Calling modules: ShowTopic
*
*   Author: Capt Frank W. DeMarco
*   History:
*   1.0 Frank W. DeMarco 8/1/85 - input original code
*****)
```

```
procedure Tframe(istart : integer);
```

```
var
```

```
    jnum : integer;
```

```
begin (* Procedure Tframe *)
```

```
    advance := 0;
```

```
    linecount := 0;
```

```
    istart := istart + 1;
```

```
    FrameHeader;
```

```
    writeln (ABLANKS,BLANKSA);
```

```
    repeat
```

```
        while (lessonln[istart,2] = '2') and (linecount < 17) do
```

```
            begin
```

```
                write ('* ');
```

```
                for jnum := 3 to 78 do
```

```
                    write (lessonln[istart,jnum]);
```

```
                writeln (' *');
```

```
                istart := istart + 1;
```

```
                linecount := linecount + 1;
```

```
            end;
```

```
    writeln (ABLANKS,BLANKSA);
```

```
    advance := 27 - (linecount + 5);
```

```

for jnum := 1 to (advance - 1) do
  writeln (ABLANKS,BLANKSA);
writeln (ALINEP1,ALINEP2);
for jnum := 1 to 27 do
  write (' ');
write ('Press RETURN to continue.');
```

readln;

linecount := 0;

if (lessonln[istart,2] = '2') then

begin

FrameHeader;

writeln (ABLANKS,BLANKSA)

end;

until (lessonln[istart,2] = '3');

if (lessonln[istart,2] = '3') then

begin

if (lessonln[istart,3] = '8') then

begin

nextframe := 0;

for jnum := 5 to 7 do

nextframe := (10 * nextframe) +

((ord(lessonln[istart,jnum]) - ord('0'))

end

else

nextframe := -1;

end;

end: (* Procedure Tframe *)

(*****)

* Date: 8/1/85 *

* Version: 1.0 *

* *

* Name: procedure Qframe *

* Module number: 6.1.5 *

* Description: Driver for the procedures that display and handle question *

* type frames. *

* Passed Variables: istart *

* Returns: None *

* Global Variables Used: lessonln *

* Global Variables Changed: None *

* Files Read: None *

* Files Written: None *

* Modules Called: ClearScreen, Mquestion, Pquestion *

* Calling modules: ShowTopic *

* *

* Author: Capt Frank W. DeMarco *

* History: *

* 1.0 Frank W. DeMarco 8/1/85 - input original code *

(*****)

procedure Qframe(istart : integer);

```

var
  qtype : char;
  ivalq : integer;

```

```

(*****
*   Date: 8/1/85
*   Version: 1.0
*
*   Name: procedure Mquestion
*   Module number: 6.1.5.1
*   Description: Displays and handles multiple choice type question frames.
*   Passed Variables:  istart
*   Returns: None
*   Global Variables Used: lessonln, test, choice, lchoice, frame, numright,
*                           nextframe
*   Global Variables Changed: numright, nextframe
*   Files Read: None
*   Files Written: temp1
*   Modules Called: FrameHeader
*   Calling modules: Qframe
*
*   Author: Capt Frank W. DeMarco
*   History:
*   1.0 Frank W. DeMarco 8/1/85 - input original code
*****)

```

```

procedure Mquestion(istart : integer);

```

```

var
  jnum : integer;
  response, correct, groupnum : char;
  mfound : boolean;

```

```

begin (* Procedure Mquestion *)

```

```

  istart := istart + 1;

```

```

  FrameHeader;

```

```

  writeln;

```

```

  writeln;

```

```

  while (lessonln[istart,2] = '2') do
    begin
      for jnum := 3 to 80 do
        write (lessonln[istart,jnum]);
      writeln;
      istart := istart + 1
    end;

```

```

  writeln;

```

```

  while (lessonln[istart,2] = '3') do

```

```

begin
  if (lessonIn[istart,4] = '+') then
    correct := lessonIn[istart,3];
    write (' ',lessonIn[istart,3],' ');
    for jnum := 6 to 80 do
      write (lessonIn[istart,jnum]);
    writeln;
    istart := istart + 1;
  end;

repeat
  writeln;
  write (' Enter your choice here ==> ');
  readln (response);
  if (response in ['A'..'E','a'..'e']) then
    writeln
  else
    writeln (' Sorry, that is not a valid response. Please try again. ');
until (response in ['A'..'E','a'..'e']);

case response of
  'a','A' : response := 'A';
  'b','B' : response := 'B';
  'c','C' : response := 'C';
  'd','D' : response := 'D';
  'e','E' : response := 'E'
end;

writeln (temp1,choice,1choice,frame:3,correct,response);

if (response = correct) then
begin
  if (test) then
    numright := numright + 1.0;
    groupnum := '4'
  end
else
  groupnum := '5';

mfound := false;
repeat
  if (lessonIn[istart,2] = groupnum) then
    mfound := true
  else
    istart := istart + 1;
until (mfound);

if (groupnum = '4') then (* Start Group '4' Logic *)
begin
  while (lessonIn[istart,4] <> 'B') or (lessonIn[istart,5] <> ':') do
    begin
      for jnum := 4 to 80 do
        write (lessonIn[istart,jnum]);

```

```

        writeln;
        istart := istart + 1
    end;
    if (lessonIn[istart,4] = 'B') and (lessonIn[istart,5] = ':') then
    begin
        nextframe := 0;
        for jnum := 6 to 8 do
            nextframe := (10 * nextframe) +
                ((ord(lessonIn[istart,jnum])) - ord('0'))
        end
    else
        nextframe := -1
    end;
    (* End Group '4' Logic *)

    if (groupnum = '5') then      (* Start Group '5' Logic *)
    begin
        if (lessonIn[istart,3] = response) and
            (lessonIn[istart,4] = ' ') then
        begin
            while (lessonIn[istart,4] <> 'B') or
                (lessonIn[istart,5] <> ':') do
            begin
                for jnum := 5 to 80 do
                    write (lessonIn[istart,jnum]);
                    writeln;
                    istart := istart + 1
                end;
            end
        else
            begin
                mfound := false;
                jnum := 3;
                repeat
                    while (lessonIn[istart,jnum] <> ' ') and not (mfound) do
                    begin
                        if (lessonIn[istart,jnum] = response) then
                            mfound := true;
                        jnum := jnum + 1
                    end;
                until (mfound) then
                begin
                    istart := istart + 1;
                    jnum := 3
                end
            else
                begin
                    while (lessonIn[istart,jnum] <> ' ') do
                        jnum := jnum + 1
                    end;
                until (mfound) or (lessonIn[istart,2] <> '5');
                if (mfound) then
                begin

```

```

        while (lessonIn[istart,4] <> 'B') or
              (lessonIn[istart,5] <> ':') do
        begin
            for jnum := jnum to 80 do
                write (lessonIn[istart,jnum]);
            writeln;
            jnum := 4;
            istart := istart + 1
        end;
        if (lessonIn[istart,4] = 'B') and (lessonIn[istart,5] = ':') then
        begin
            nextframe := 0;
            for jnum := 6 to 8 do
                nextframe := (10 * nextframe) +
                             ((ord(lessonIn[istart,jnum])) - ord('0'))
            end
        else
            nextframe := -1
        end
        else
            writeln ('SOMETHING IS AWRY! LET'S TRY THAT AGAIN.')
        end;
    end;                                     (* End Group '5' Logic *)

end;                                     (* Procedure Mquestion *)

(*****
*   Date: 9/1/85
*   Version: 1.0
*
*   Name: procedure Pquestion
*   Module number: 5.1.5.2
*   Description: Displays and handles pick type question frames (true/false
*               and yes/no).
*   Passed Variables: istart
*   Returns: None
*   Global Variables Used: lessonIn, choice, lchoice, frame, test, numright,
*               nextframe
*   Global Variables Changed: numright, nextframe
*   Files Read: None
*   Files Written: temp1
*   Modules Called: FrameHeader
*   Calling modules: Qframe
*
*   Author: Capt Frank W. DeMarco
*   History:
*       1.0 Frank W. DeMarco 8/1/85 - input original code
*****)

procedure Pquestion(istart : integer);

type
    sresponses = packed array [1..5] of char;

```

```

var
  jnum, index : integer;
  answer : sresponses;
  correct, response, groupnum : char;
  pfound, ptrue : boolean;

begin  (* Procedure Pquestion *)

  istart := istart + 1;
  FrameHeader;
  writeln;

  while (lessonln[istart,2] = '2') do
    begin
      for jnum := 3 to 80 do
        write (lessonln[istart,jnum]);
        writeln;
        istart := istart + 1
      end;

      writeln;

      if (lessonln[istart,2] = '3') then
        begin
          correct := lessonln[istart,3];
          istart := istart + 1
        end;

      repeat
        index := 1;
        write ('Enter your choice here ==> ');
        while not (eoln) and (index < 6) do
          begin
            read (answer[index]);
            index := index + 1
          end;
        if (eoln) and (i < 6) then
          begin
            for index := index to 5 do
              answer[index] := ' ';
            readln;
          end
        else
          readln;
        ptrue := false;

        if (answer[1] in ['t','T','f','F','y','Y','n','N']) then
          ptrue := true
        else
          writeln ('Sorry, that is not a valid response. Please try again.');
```

```

until (ptrue);
```



```

case answer[1] of
  't','T','y','Y' : response := 'Y';
  'f','F','n','N' : response := 'N';
end;

writeln (temp1,choice,1choice,frame:3,correct,response);

if (response = correct) then
begin
  if (test) then
    numright := numright + 1.0;
  groupnum := '4';
end
else
  groupnum := '5';

pfound := false;

repeat
  if (lessonln[istart,2] = groupnum) then
    pfound := true
  else
    istart := istart + 1;
until (pfound);

while (lessonln[istart,4] <> 'B') or (lessonln[istart,5] <> ':') do
begin
  writeln;
  for jnum := 4 to 80 do
    write (lessonln[istart,jnum]);
    istart := istart + 1;
end;

if (lessonln[istart,4] = 'B') and (lessonln[istart,5] = ':') then
begin
  nextframe := 0;
  for jnum := 6 to 8 do
    nextframe := (10 * nextframe) +
      ((ord(lessonln[istart,jnum])) - ord('0'));
end
else
  nextframe := -1;

end;  (* Procedure Pquestion *)

(*****
(* Start of main part of procedure: Dframe *)
*****)

begin  (* Procedure Dframe *)

  if (lessonln[istart,14] = 'M') then
    atype := 'M'

```

```

else
  qtype := 'P';

case qtype of
  'M' : Mquestion(istart);
  'P' : Pquestion(istart)
end;

writeln;
for ivalq := 1 to 27 do
  write (' ');
write ('Press RETURN to continue. ');
readln;

end; (* Procedure Qframe *)

(*****)
(* Start of main part of procedure: ShowTopic *)
(*****)

begin (* Procedure ShowTopic *)

  BlankLines;
  ReadLines;
  StorePositions;

  numasked := 0.0;
  numright := 0.0;
  score := 0.0;
  test := false;
  istart := 1;

  frame := 0;
  for j := 9 to 11 do
    frame := (10 * frame) + ((ord(lessonln[istart,j]) - ord('0')));

  if (lessonln[istart,13] = 'T') and (lessonln[istart,14] = 'T') then
    test := true;

  repeat
    if (lessonln[istart,2] = '1') then
      begin
        ftype := lessonln[istart,13];
        if (test) and (ftype = '0') then
          numasked := numasked + 1.0;
        case ftype of
          'T' : Tframe(istart);
          '0' : Qframe(istart)
        end;
        k := 0;
        sfound := false;

```

```

repeat
  k := k + 1;
  if (lplace[k].framenum = nextframe) then
    begin
      istart := lplace[k].ivalue;
      frame := lplace[k].framenum;
      sfound := true
    end;
  until (sfound) or (lplace[k].framenum = -1);
end;

until not (sfound) or (lplace[k].framenum = -1);

i := ord(choice) - ord('0');
j := ord(lchoice) - ord('0');

if (test) then
  begin
    ClearScreen;
    score := numright/nummasked;
    score := (score * 100.0);
    writeln ('Your test score = ',score:-1:2,'%');
    writeln;
    if (score >= MINSORE) then
      begin
        writeln ('CONGRATULATIONS! YOU HAVE PASSED THE LESSON TEST. ');
        npupil.topics[i,j] := '+';
        npupil.lessons[i] := '+';
        topicstat[j] := '+'
      end
    else
      begin
        writeln ('Sorry, but you missed too many questions to pass the test. ');
        writeln;
        write ('I suggest that you review at least one topic before you ');
        writeln ('retake the lesson test. ');
        npupil.topics[i,j] := '-';
        npupil.lessons[i] := '-';
        topicstat[j] := '-'
      end
    end
  else
    begin
      ClearScreen;
      npupil.topics[i,j] := '+';
      topicstat[j] := '+'
    end;
end;

end; (* Procedure ShowTopic *)

```

```

(*****
*   Date: 8/1/85                                     *
*   Version: 1.0                                     *

```

```

*
* Name: procedure RecordStats
* Module number: 6.2
* Description: Reads file "STATS" and adds statistical data from current
*              session.
* Passed Variables: None
* Returns: None
* Global Variables Used: println
* Global Variables Changed: println
* Files Read: stats, temp1
* Files Written: temp2, stats
* Modules Called: None
* Calling modules: StartLesson
*
* Author: Capt Frank W. DeMarco
* History:
* 1.0 Frank W. DeMarco 8/1/85 - input original code
*****}

```

```

procedure RecordStats;

```

```

var

```

```

    i : integer;

```

```

begin (* Procedure RecordStats *)

```

```

    writeln;

```

```

    writeln ('One moment please, while I update my records.');
```

```

    assign (temp2,'t2');
```

```

    assign (statfile,'stats');
```

```

    reset (temp1);
```

```

    rewrite (temp2);
```

```

    reset (statfile);

```

```

repeat

```

```

    while not (eof(statfile)) do

```

```

        begin

```

```

            for i := 1 to 80 do

```

```

                println[i] := ' ';

```

```

            readln (statfile,println);

```

```

            writeln (temp2,println)

```

```

        end;

```

```

until (eof(statfile));

```

```

repeat

```

```

    while not (eof(temp1)) do

```

```

        begin

```

```

            for i := 1 to 80 do

```

```

                println[i] := ' ';

```

```

            readln (temp1,println);

```

```

            writeln (temp2,println)

```

```

        end;

```

```

until (eof(temp1));

```

```

reset (temp2);
rewrite (statfile);

repeat
  while not (eof(temp2)) do
    begin
      for i := 1 to 80 do
        println[i] := ' ';
      readln (temp2,println);
      writeln (statfile,println)
    end;
until (eof(temp2));

rewrite (temp1);
rewrite (temp2);
close (temp1);
close (temp2);
close (statfile);

end;  (* Procedure RecordStats *)

(*****
*   Date: 8/1/85
*   Version: 1.0
*
*   Name: procedure StuRec
*   Module number: 6.3
*   Description: Writes updated student course progress data to file
*                 "STUDENT".
*   Passed Variables: None
*   Returns: None
*   Global Variables Used: studentcount, rpupil, npupil
*   Global Variables Changed: none
*   Files Read: None
*   Files Written: student
*   Modules Called: None
*   Calling modules: StartLesson
*
*   Author: Capt Frank W. DeMarco
*   History:
*     1.0 Frank W. DeMarco 8/1/85 - input original code
*****)

procedure StuRec;

var
  i, ii, j : integer;

begin  (* Procedure StuRec *)

  rewrite (student);

```

```

for i := 1 to studentcount do
  if (rpupil[i].studentnumber <> npupil.studentnumber) and
    (rpupil[i].studentnumber <> ' ') then
    begin
      write(student,'>');
      for j := 1 to 11 do
        write(student,rpupil[i].studentnumber[j]);
      for j := 1 to 28 do
        write(student,rpupil[i].studentname[j]);
      for j := 1 to MAXLESSONS do
        write(student,rpupil[i].lessons[j]);
      for ii := 1 to MAXLESSONS do
        begin
          if (ii < MAXLESSONS) then
            for j := 1 to MAXTOPICS do
              write (student,rpupil[i].topics[ii,j])
            else
              begin
                for j := 1 to (MAXTOPICS-1) do
                  write (student,rpupil[i].topics[ii,j]);
                writeln (student,rpupil[i].topics[ii,MAXTOPICS])
              end;
            end;
        end;
      end;

      write(student,'>');
      for i := 1 to 11 do
        write(student,npupil.studentnumber[i]);
      for i := 1 to 28 do
        write(student,npupil.studentname[i]);
      for i := 1 to MAXLESSONS do
        write(student,npupil.lessons[i]);
      for i := 1 to MAXLESSONS do
        begin
          for j := 1 to MAXTOPICS do
            write (student,npupil.topics[i,j])
          end;
        end;

      close (student)

end;  (* Procedure StuRec *)

(*****
*   Date: 8/1/85
*   Version: 1.0
*
*   Name: procedure StartLesson
*   Module number: 6.0
*   Description: Displays topic choices for a lesson, prompts student for
*                 choice of topic to be shown. Driver of procedures that
*                 display lesson material and update statistical & student
*                 progress files.
*   Passed Variables: None
*
*****

```

```

* Returns: None
* Global Variables Used: choice, topicstat, linecount, println, advance,
*                        menuIn, lchoice
* Global Variables Changed: topicstat, linecount, println, advance, menuIn,
*                        lchoice
* Files Read: None
* Files Written: None
* Modules Called: ClearScreen, ShowTopic, RecordStats, StuRec
* Calling modules: program CAI
*
* Author: Capt Frank W. DeMarco
* History:
* 1.0 Frank W. DeMarco 8/1/85 - input original code
*****}

```

```

procedure StartLesson:

```

```

var

```

```

    i, j, k, index : integer;
    character : char;

```

```

begin (* Procedure StartLesson *)

```

```

    case choice of

```

```

        '1' : assign (lesson,'b:lesson1');
        '2' : assign (lesson,'b:lesson2');
        '3' : assign (lesson,'b:lesson3');
        '4' : assign (lesson,'b:lesson4');
        '5' : assign (lesson,'b:lesson5');
        '6' : assign (lesson,'b:lesson6');
    end;

```

```

    index := ord(choice) - ord('0');
    for i := 1 to MAXTOPICS do
        topicstat[i] := npupil.topics[index,i];

```

```

    reset (lesson);
    read (lesson,character);
    linecount := 0;

```

```

    repeat

```

```

    while (character = '#') and (linecount < 23) do
        begin
            for i := 1 to 80 do
                println[i] := ' ';
            readln (lesson,println);
            linecount := linecount + 1;
            writeln (println);
            read (lesson,character)
        end;

```

```

if (character = '!') then
begin
  advance := 23 - linecount;
  linecount := 0;
  for i := 1 to advance do
    writeln;
  for i := 1 to 27 do
    write (' ');
  write ('Press RETURN to continue.');
```

readln;

```

  readln (lesson);
  read (lesson,character);
  ClearScreen
end;
```

until not (character in ['#','!']);

```

for i := 1 to 22 do
begin
  for j := 1 to 80 do
    menuIn[i,j] := ' ';
  end;
```

menuIn[i,1] := character;

```

for j := 2 to 79 do
  read (lesson,menuIn[i,j]);
readln (lesson,menuIn[i,80]);
```

```

for i := 2 to 22 do
begin
  for j := 1 to 79 do
    read (lesson,menuIn[i,j]);
  readln (lesson,menuIn[i,80])
end;
```

repeat

```

  assign (temp1,'t1');
  rewrite (temp1);
  repeat
    k := 0;
    i := 1;
    while (menuIn[i,1] in ['*','@']) do
      begin
        if (menuIn[i,1] = '*') then
          begin
            for j := 1 to 79 do
              write (menuIn[i,j]);
            writeln (menuIn[i,80])
          end
        else
          begin
            k := k + 1;
            write ('*');
```



```

        for j := 2 to 8 do
            write (menuIn[i,j]);
        write (topicstat[k]);
        for j := 10 to 79 do
            write (menuIn[i,j]);
        writeln (menuIn[i,80])
    end;
    i := i + 1
end;

writeln;
write ('ENTER THE TOPIC NUMBER OF YOUR CHOICE OR "X" TO EXIT THIS LESSON: ');
readln (lchoice);
if (lchoice in ['1'..VTOPIC,'x','X']) then
    ClearScreen
else
    begin
        ClearScreen;
        writeln ('Sorry, ',lchoice,' is not a valid response. Please try again.')
    end;

until (lchoice in ['1'..VTOPIC,'x','X']);

if (lchoice in ['1'..VTOPIC]) then
    writeln ('You have chosen topic number ',lchoice,'. Thank you.')
else
    writeln ('OK, I will now return you to the lesson selection menu.');
```

```

if (lchoice in ['1'..VTOPIC]) then
    begin
        ShowTopic;
        RecordStats;
        StuRec;
        ClearScreen
    end;

until (lchoice in ['x','X']);

close (temp1);
close (lesson)

end; (* Procedure StartLesson *)

(*****
(* Start of main driver: Program CAI *)
(*****

begin (* Program CAI *)

    ClearScreen;
    StartEnd('S');
    Query;
```

```

if (studentcount <= MAXSTUDENTS) then
begin
  repeat
    begin
      Select;
      if (choice in ['1'..VLESSON]) then
        StartLesson
      end
    until (choice in ['x','X']);
    StartEnd('E')
  end
end.  (* Program CAI *)

```

Program "STUDENT STATUS"

(**** THIS PROGRAM WAS WRITTEN IN PARTIAL FULFILLMENT OF A MASTERS THESIS ****)

(*****)

```
*   Date: 8/1/85                                           *
*   Version: 1.0                                           *
*                                                         *
*   Title: Program Student_Status                         *
*   Filename: STATUS.PAS                                   *
*   Coordinator: Capt Frank W. DeMarco                    *
*   Project: Masters Thesis                               *
*   Operating System: MS-DOS                              *
*   Language: Pascal                                     *
*   Use: Compile and link with PASCAL.LIB using MS-Pascal compiler and linker.*
*   Contents: Program Student_Status - Main Driver.      *
*               Procedure ClearScreen - Clears 2-100 terminal screen.      *
*               Procedure QueryUser - Determines the users preferred method of *
*                   program output (screen or hardcopy).    *
*               Procedure Header - Produces the program report header.      *
*               Procedure Display - Produces the status report for all students *
*                   in file "STUDENTS".                      *
*               Procedure EndScreen - Completes the screen display format for *
*                   the screen method of program output.    *
*                                                         *
*   Function: The purpose of this program is to provide a means for training *
*               managers, as well as personnel of the CAI Plans Branch (3300 *
*               TCHTW at Keesler AFB, to check student progress in the C CAI *
*               course.                                     *
*                                                         *
*****)
```

(*****)

```
*   Date: 8/1/85                                           *
*   Version: 1.0                                           *
*                                                         *
*   Name: program Student_Status                          *
*   Module number: 1.0                                     *
*   Description: Main driver of program                    *
*   Passed Variables: None                                 *
*   Returns: None                                          *
*   Global Variables Used: studentcount, character, advance, choice          *
*   Global Variables Changed: studentcount, character, advance                *
*   Files Read: student                                   *
*   Files Written: None                                    *
*   Modules Called: ClearScreen, QueryUser, Header, Display, EndScreen        *
*   Calling modules: None                                  *
*                                                         *
*   Author: Capt Frank W. DeMarco                         *
*   History:                                               *
*       1.0 Frank W. DeMarco 8/1/85 - input original code *
*****)
```

```
program Student_Status (input,output);
```

```
const
```

```
  ALINEP1 = '*****';
```

```
  ALINEP2 = '*****';
```

```
  ADASH = '*-----';
```

```
  DASHA = '-----*';
```

```
  NUMLESSONS = 6;
```

```
  NUMTOPICS = 30;
```

```
type
```

```
  iofile = TEXT;
```

```
var
```

```
  infile : iofile;
```

```
  choice, character : char;
```

```
  advance, i, studentcount : integer;
```

```
{*****}
```

```
*   Date:  9/1/85                                           *
```

```
*   Version: 1.0                                           *
```

```
*                                           *
```

```
*   Name: procedure ClearScreen                           *
```

```
*   Module number: 1.1                                     *
```

```
*   Description: Clears Z-100 terminal screen and sets "no-wrap" on EOL. *
```

```
*   Passed Variables: None                                *
```

```
*   Returns: None                                          *
```

```
*   Global Variables Used: None                           *
```

```
*   Global Variables Changed: None                        *
```

```
*   Files Read: None                                       *
```

```
*   Files Written: None                                    *
```

```
*   Modules Called: None                                   *
```

```
*   Calling modules: program Student_Status               *
```

```
*                                           *
```

```
*   Author: Capt Frank W. DeMarco                         *
```

```
*   History:                                               *
```

```
*       1.0 Frank W. DeMarco 9/1/85 - input original code *
```

```
{*****}
```

```
procedure ClearScreen;
```

```
begin (* Procedure ClearScreen *)
```

```
  write (chr(27), 'H', chr(27), 'J', chr(27), 'w')
```

```
end; (* Procedure ClearScreen *)
```

```
{*****}
```

```
*   Date: 9/1/85                                           *
```

```
*   Version: 1.0                                           *
```

```
*                                           *
```

```
*   Name: procedure QueryUser                             *
```

```

*   Module number: 1.0
*   Description: Determines the users preferred method of program output
*               (screen or hardcopy).
*   Passed Variables: choice
*   Returns: choice
*   Global Variables Used: None
*   Global Variables Changed: None
*   Files Read: None
*   Files Written: None
*   Modules Called: None
*   Calling modules: program Student_Status
*
*   Author: Capt Frank W. DeMarco
*   History:
*   1.0 Frank W. DeMarco 8/1/85 - input original code
*****

```

```

procedure QueryUser (var choice: char):

```

```

begin (* Procedure QueryUser *)

```

```

    write ('The output of this program can be put into two (2) different');
    writeln (' formats. ');
    writeln;
    writeln ('If you plan on getting a hard copy; type  H  . ');
    writeln ('If you only want a screen display ; type  S  . ');

```

```

    repeat
        writeln;
        write ('Enter your choice here ==> ');
        readln (choice);
    until (choice in ['h', 'H', 's', 'S']);

```

```

    if (choice in ['h', 'H']) then
        begin
            writeln;
            writeln ('Press  ^P  (CONTROL P) and then  RETURN  to get printout');
            readln
        end

```

```

end; (* Procedure QueryUser *)

```

```

(*****
*   Date: 8/1/85
*   Version: 1.0
*
*   Name: procedure Header
*   Module number: 1.0
*   Description: Produces the program report header.
*   Passed Variables: None
*   Returns: None
*   Global Variables Used: None
*   Global Variables Changed: None
*)

```

```

*   Files Read: None
*   Files Written: None
*   Modules Called: None
*   Calling modules: program Student_Status, EndScreen
*
*   Author: Capt Frank W. DeMarco
*   History:
*   1.0 Frank W. DeMarco 8/1/85 - input original code
*****

```

```

procedure Header;

```

```

begin (* Procedure Header *)

```

```

    writeln (ALINEP1,ALINEP2);
    write ('*   THE FOLLOWING IS THE PRESENT STUDENT STATUS FOR STUDENTS ON');
    writeln (' THIS DISK.   *');
    writeln (ALINEP1,ALINEP2);
    write ('*   STUDENT   : LESSON   : LESSON   : LESSON   : LESSON   :');
    writeln (' LESSON   : LESSON *');
    write ('*   ID #       :   #1       :   #2       :   #3       :   #4       :');
    writeln ('       #5       :   #6   *');
    writeln (ADASH,DASHA)

```

```

end: (* Procedure Header *)

```

```

(*****
*   Date: 8/1/85
*   Version: 1.0
*
*   Name: procedure Display
*   Module number: 1.4
*   Description: Produces the status report for all students in file
*               "STUDENTS".
*   Passed Variables: character, studentcount
*   Returns: character, studentcount
*   Global Variables Used: None
*   Global Variables Changed: None
*   Files Read: student
*   Files Written: None
*   Modules Called: None
*   Calling modules: program Student_Status
*
*   Author: Capt Frank W. DeMarco
*   History:
*   1.0 Frank W. DeMarco 8/1/85 - input original code
*****)

```

```

procedure Display(var character: char;
                  var studentcount: integer);

```

```

type
    roll = record

```

```

    studentnumber : packed array [1..11] of char;
    studentname : packed array [1..28] of char;
    lessons : packed array [1..NUMLESSONS] of char;
    topics : packed array [1..NUMTOPICS] of char;
end;

var
    pupil : roll;

begin (* Procedure Display *)

    studentcount := studentcount + 1;
    while not (eof(infile)) do
        begin
            for i := 1 to 11 do
                read (infile,pupil.studentnumber[i]);
            for i := 1 to 28 do
                read (infile,pupil.studentname[i]);
            for i := 1 to NUMLESSONS do
                read (infile,pupil.lessons[i]);
            for i := 1 to NUMTOPICS do
                read (infile,pupil.topics[i]);
            end;

            write ('* ');
            for i := 1 to 11 do
                begin
                    if (pupil.studentnumber[i] = '*') then
                        write (' ')
                    else
                        write (pupil.studentnumber[i])
                    end;
                write (' ');
            for i := 1 to NUMLESSONS do
                begin
                    if (pupil.lessons[i] = '+') then
                        write (' Passed ')
                    else
                        begin
                            if (i < NUMLESSONS) then
                                write (' ')
                            else
                                write (' ')
                            end;
                        if (i < NUMLESSONS) then
                            write ('')
                        else
                            writeln ('*');
                        end;
                    end;
                if not (eof(infile)) then
                    readln (infile);

```

```

if not (eof(infile)) then
  read (infile,character);

```

```

end;    (* Procedure Display *)

```

```

(*****
*   Date: 9/1/85
*   Version: 1.0
*
*   Name: procedure EndScreen
*   Module number: 1.5
*   Description: Completes the screen display format for the screen method of
*                 program output.
*   Passed Variables: advance
*   Returns: None
*   Global Variables Used: None
*   Global Variables Changed: None
*   Files Read: None
*   Files Written: None
*   Modules Called: ClearScreen
*   Calling modules: program Student_Status
*
*   Author: Capt Frank W. DeMarco
*   History:
*     1.0 Frank W. DeMarco 9/1/85 - input original code
*****)

```

```

procedure EndScreen(advance: integer);

```

```

begin    (* Procedure EndScreen *)

```

```

  for i := 1 to (advance - 1) do
    begin
      write ('*           |           |           |');
      writeln ('           |           |           *');
    end;
    writeln (ALINEP1,ALINEP2);
    for i := 1 to 24 do
      write (' ');
    if not (eof(infile)) then
      begin
        write ('Press RETURN to continue. ');
        readln;
        ClearScreen;
        Header;
      end
    else
      begin
        write ('Press RETURN to end program. ');
        readln;
      end
    end;

```

```

end;    (* Procedure EndScreen *)

```



```

(*****)
(* Start of main driver:  Program Student_Status  *)
(*****)

```

```

begin  (* Program Student_Status *)

```

```

  ClearScreen;
  QueryUser(choice);
  ClearScreen;
  Header;

```

```

  assign (infile,'student');
  reset (infile);
  read (infile,character);

```

```

  repeat
    studentcount := 0;
    while (character = '>>') and (studentcount < 16) and not (eof(infile)) do
      Display(character,studentcount);
      advance := 23 - (studentcount + 6);
      if (choice in ['s','S']) then
        EndScreen(advance);
    until (eof(infile));

```

```

  if (choice in ['s','S']) then
    ClearScreen
  else
    writeln (ALINEP1,ALINEP2)

```

```

end.  (* Program Student_Status *)

```

Program "CAI STATISTICS"

(**** THIS PROGRAM WAS WRITTEN IN PARTIAL FULFILLMENT OF A MASTERS THESIS *****)

(*****)

* Date: 10/15/85 *

* Version: 1.0 *

* *

* Title: Program CAI_Statistics *

* Filename: VALIDATE.PAS *

* Coordinator: Capt Frank W. DeMarco *

* Project: Masters Thesis *

* Operating System: MS-DOS *

* Language: Pascal *

* Use: Compile and link with PASCAL.LIB using MS-Pascal compiler and linker.*

* Contents: PROGRAM CAI_Statistics - Main Driver *

* Procedure ClearScreen - Clears Z-100 terminal screen. *

* Procedure QueryUser - Determines the users preferred method of *

* program output (screen or hardcopy). *

* Procedure Header - Produces the program report header. *

* Procedure Init - Initializes array and two link lists used in *

* the program as well as opening file "STATS". *

* Procedure ReadStats - Reads file "STATS" into a linked list of *

* frame records as well as builds an array *

* of unique frame identifiers. *

* Procedure Sort - Sorts the frame identifier array into numeric *

* order. *

* Procedure Display - Driver for the procedures that display the *

* statistics for each unique question frame. *

* Procedure FinalScreen - Wraps up the screen display after all *

* statistics have been processed. *

* Procedure BuildFrameLL - Constructs a linked list of frame re- *

* cords that are of the same frame. *

* Procedure InitDisplay - Initilizes variables used in statistical*

* analysis. *

* Procedure ShowStats - Analyzes and displays statistical data *

* stored in the linked list of frame records*

* (of the same frame). *

* Procedure EndScreen - Wraps up the screen display after there *

* has been a full screen displayed. *

* *

* Function: The purpose of this program is to provide a means for the OPR *

* at Kessler AFB, to validate course material and teaching effec- *

* tiveness by analyzing questions asked during lesson and test *

* course presentation. *

* *

(*****)

(*****)

* Date: 10/15/85 *

* Version: 1.0 *

* *

* Name: program CAI_Statistics *

* Module number: 1.0 *


```

(*****
*   Date: 10/15/85
*   Version: 1.0
*
*   Name: ClearScreen
*   Module number: 1.1
*   Description: Clears Z-100 terminal screen and sets "no-wrap" on EOL.
*   Passed Variables: None
*   Returns: None
*   Global Variables Used: None
*   Global Variables Changed: None
*   Files Read: None
*   Files Written: None
*   Modules Called: None
*   Calling modules: program CAI_Statistics, FinalScreen, EndScreen
*
*   Author: Capt Frank W. DeMarco
*   History:
*       1.0 Frank W. DeMarco 10/15/85 - input original code
*****)

```

procedure ClearScreen;

begin (* Procedure ClearScreen *)

 write (chr(27),'H',chr(27),'J',chr(27),'w');

end; (* Procedure ClearScreen *)

```

(*****
*   Date: 10/15/85
*   Version: 1.0
*
*   Name: QueryUser
*   Module number: 1.2
*   Description: Determines the users preferred method of program output
*               (screen or hardcopy).
*   Passed Variables: choice
*   Returns: choice
*   Global Variables Used: None
*   Global Variables Changed: None
*   Files Read: None
*   Files Written: None
*   Modules Called: None
*   Calling modules: program CAI_Statistics
*
*   Author: Capt Frank W. DeMarco
*   History:
*       1.0 Frank W. DeMarco 10/15/85 - input original code
*****)

```

```

procedure QueryUser(var choice: char);

begin  (* Procedure QueryUser *)

    write ('The output of this program can be put into two (2) different');
    writeln (' formats. ');
    writeln;
    writeln ('If you plan on getting a hard copy; type  H  . ');
    writeln ('If you only want a screen display ; type  S  . ');

    repeat
        writeln;
        write ('Enter your choice here ==> ');
        readln (choice);
    until (choice in ['h', 'H', 's', 'S']);

    if (choice in ['h', 'H']) then
        begin
            writeln;
            writeln ('Press  ^P  (CONTROL P) and then  RETURN  to get printout');
            readln;
        end;

end;  (* Procedure QueryUser *)

(*****
*   Date: 10/15/85
*   Version: 1.0
*
*   Name: Header
*   Module number: 1.3
*   Description: Produces the program report header.
*   Passed Variables: None
*   Returns: None
*   Global Variables Used: None
*   Global Variables Changed: None
*   Files Read: None
*   Files Written: None
*   Modules Called: None
*   Calling modules: program CAI_Statistics, EndScreen
*
*   Author: Capt Frank W. DeMarco
*   History:
*       1.0 Frank W. DeMarco 10/15/85 - input original code
*****)

procedure Header;

begin  (* Procedure Header *)

    writeln (ALINEP1, ALINEP2);
    write ('*   THE FOLLOWING IS A STATISTICAL VALIDATION REPORT FOR THE C ');
    writeln ('CAI COURSE.   *');

```

```

writeln (ALINEP1,ALINEP2);
write ('* L # I F # I # A I # B I # C I # D I # E I # Y I # N *');
writeln (' # R I # W I % R I % W *');
writeln (ADASH,DASHA)
end;    (* Procedure Header *)

```

```

(*****
*   Date: 10/15/85
*   Version: 1.0
*
*   Name: Init
*   Module number: 1.4
*   Description: Initializes array and two link lists used in the program as
*                well as opening file "STATS".
*   Passed Variables: None
*   Returns: None
*   Global Variables Used: stats, filehead, filenode, node, framehead,
*                          framemode, character
*   Global Variables Changed: stats, filehead, filenode, framehead, framemode
*                          character
*   Files Read: stats
*   Files Written: None
*   Modules Called: None
*   Calling modules: program CAI_Statistics
*
*   Author: Capt Frank W. DeMarco
*   History:
*   1.0 Frank W. DeMarco 10/15/85 - input original code
*****)

```

```

procedure Init;

```

```

var
  i : integer;

```

```

begin (* Procedure Init *)
  for i := 1 to 150 do
    stats[i] := 0;

```

```

    filehead := nil;
    new (node);
    filenode := node;
    filehead := filenode;

```

```

    framehead := nil;
    new (node);
    framemode := node;
    framehead := framemode;

```

```

    assign (infile,'stats');
    reset (infile);
    read (infile,character);
end;    (* Procedure Init *)

```

```

*****
*   Date: 10/15/85                                     *
*   Version: 1.0                                       *
*   *                                                 *
*   Name: ReadStats                                   *
*   Module number: 1.5                                 *
*   Description: Reads file "STATS" into a linked list of frame records as *
*               well as builds an array of unique frame identifiers.      *
*   Passed Variables: None                             *
*   Returns: None                                     *
*   Global Variables Used: tempbuff, stats, node, filenode, character      *
*   Global Variables Changed: tempbuff, stats, filenode, character        *
*   Files Read: stats                                  *
*   Files Written: None                               *
*   Modules Called: None                              *
*   Calling modules: program CAI_Statistics            *
*   *                                                 *
*   Author: Capt Frank W. DeMarco                     *
*   History:                                           *
*   1.0 Frank W. DeMarco 10/15/85 - input original code *
*****

```

```

procedure ReadStats;

```

```

var

```

```

    i : integer;
    fnumber : integer4;
    inchar : char;
    found : boolean;

```

```

begin (* Procedure ReadStats *)

```

```

    fnumber := 0;
    fnumber := (10 * fnumber) + ((ord(character)) - ord('0'));
    for i := 1 to 4 do
        begin
            read (infile,inchar);
            fnumber := (10 * fnumber) + ((ord(inchar)) - ord('0'))
        end;

```

```

    temptbuff.ltframe_num := fnumber;
    read (infile,temptbuff.c_answer);
    read (infile,temptbuff.e_response);
    i := 1;
    found := false;
    while (stats[i] <> 0) do
        begin
            if (stats[i] = temptbuff.ltframe_num) then
                found := true;
            i := i+1
        end;

```

```

if not (found) then
    stats[i] := tempbuff.ltframe_num;
new (node);
filenode^.next := node;
filenode      := node;
filenode^.ltf_num := tempbuff.ltframe_num;
filenode^.c_ans := tempbuff.c_answer;
filenode^.s_ans := tempbuff.s_response;
filenode^.next := nil;
if not (eof(infile)) then
    readln (infile);
if not (eof(infile)) then
    read (infile,character);

```

```

end;    (* Procedure ReadStats *)

```

```

(*****
*   Date: 10/15/85                                     *
*   Version: 1.0                                       *
*                                                       *
*   Name: Sort                                         *
*   Module number: 1.6                                 *
*   Description: Sorts the frame identifier array into numeric order. *
*   Passed Variables: stats                           *
*   Returns: stats                                    *
*   Global Variables Used: None                       *
*   Global Variables Changed: None                   *
*   Files Read: None                                  *
*   Files Written: None                               *
*   Modules Called: None                              *
*   Calling modules: program CAI_Statistics           *
*                                                       *
*   Author: Capt Frank W. DeMarco                     *
*   History:                                           *
*   1.0 Frank W. DeMarco 10/15/85 - input original code *
*****)

```

```

procedure Sort(var stats: stats_array);

```

```

var
    temp : integer4;
    sindx, ival, imax : integer;

```

```

begin    (* Procedure Sort *)

```

```

    imax := 0;
    sindx := 1;
    while (stats[sindx] <> 0) do
        begin
            imax := imax + 1;
            sindx := sindx + 1;
        end;

```



```

repeat
  sindx := 1;
  for ival := 1 to (imax-1) do
    begin
      temp := stats[sindx];
      if (temp > stats[sindx+1]) then
        begin
          stats[sindx] := stats[sindx+1];
          stats[sindx+1] := temp;
        end;
      sindx := sindx + 1;
    end;
  imax := (sindx-1);
until (imax = 0);

```

end; (* Procedure Sort *)

```

(*****
*   Date: 10/15/85
*   Version: 1.0
*
*   Name: Display
*   Module number: 1.7
*   Description: Driver for the procedures that display the statistics for
*               each unique question frame.
*   Passed Variables: None
*   Returns: None
*   Global Variables Used: filenode, filehead, framenode, framehead
*   Global Variables Changed: filenode, framenode
*   Files Read: None
*   Files Written: None
*   Modules Called: BuildFrameLL, InitDisplay, ShowStats
*   Calling modules: program CAI_Statistics
*
*   Author: Capt Frank W. DeMarco
*   History:
*   1.0 Frank W. DeMarco 10/15/85 - input original code
*****)

```

procedure Display;

```

(*****
*   Date: 10/15/85
*   Version: 1.0
*
*   Name: BuildFrameLL
*   Module number: 1.7.1
*   Description: Constructs a linked list of frame records that are of the
*               same frame.
*   Passed Variables: None
*   Returns: None
*   Global Variables Used: filenode, stats, node, filenode
*   Global Variables Changed: framenode, filenode

```

```

*   Files Read: None
*   Files Written: None
*   Modules Called: None
*   Calling modules: Display
*
*   Author: Capt Frank W. DeMarco
*   History:
*   1.0 Frank W. DeMarco 10/15/85 - input original code
*****

```

```

procedure BuildFrameLL;

```

```

begin (* Procedure BuildFrameLL *)

```

```

  if (filenode^.ltf_num = stats[dindx]) then
    begin
      new (node);
      framenode^.next := node;
      framenode       := node;
      framenode^.ltf_num := filenode^.ltf_num;
      framenode^.c_ans := filenode^.c_ans;
      framenode^.s_ans := filenode^.s_ans;
      framenode^.next := nil
    end;
  filenode := filenode^.next;

```

```

end; (* Procedure BuildFrameLL *)

```

```

(*****
*   Date: 10/15/85
*   Version: 1.0
*
*   Name: InitDisplay
*   Module number: 1.7.2
*   Description: Initializes variables used in statistical analysis.
*   Passed Variables: None
*   Returns: None
*   Global Variables Used: advance, num_seen, num_right, num_wrong,
*                           percent_right, percent_wrong, num_A, num_B, num_C
*                           num_D, num_E, num_Y, num_N
*   Global Variables Changed: advance, num_seen, num_right, num_wrong,
*                              percent_right, percent_wrong, num_A, num_B, num_C
*                              num_D, num_E, num_Y, num_N
*   Files Read: None
*   Files Written: None
*   Modules Called: None
*   Calling modules: Display
*
*   Author: Capt Frank W. DeMarco
*   History:
*   1.0 Frank W. DeMarco 10/15/85 - input original code
*****

```

```

procedure InitDisplay;

begin  (* Procedure InitDisplay *)

  advance      := 0;
  num_seen     := 0.0;
  num_right    := 0.0;
  num_wrong    := 0.0;
  percent_right := 0.0;
  percent_wrong := 0.0;
  num_A := 0;
  num_B := 0;
  num_C := 0;
  num_D := 0;
  num_E := 0;
  num_Y := 0;
  num_N := 0;

end;  (* Procedure InitDisplay *)

{*****}
*   Date: 10/15/85                                     *
*   Version: 1.0                                         *
*                                                         *
*   Name: ShowStats                                     *
*   Module number: 1.7.3                                 *
*   Description: Analyzes and displays statistical data stored in the linked *
*                 list of frame records (of the same frame). *
*   Passed Variables: None                               *
*   Returns: None                                        *
*   Global Variables Used: framenode, num_right, num_wrong, num_seen, num_A *
*                           num_B, num_C, num_D, num_E, num_Y, num_N, tot_r *
*                           tot_w, percent_right, percent_wrong, linecount *
*                           advance *
*   Global Variables Changed: framenode, num_right, num_wrong, num_seen, num_A *
*                           num_B, num_C, num_D, num_E, num_Y, num_N, tot_r *
*                           tot_w, percent_right, percent_wrong, linecount *
*                           advance *
*   Files Read: None                                     *
*   Files Written: None                                  *
*   Modules Called: EndScreen                            *
*   Calling modules: Display                             *
*                                                         *
*   Author: Capt Frank W. DeMarco                       *
*   History: *
*     1.0 Frank W. DeMarco 10/15/85 - input original code *
{*****}

procedure ShowStats;

var
  lesson_number, frame_number : integer4;

```

```

(*****
*   Date: 10/15/85
*   Version: 1.0
*
*   Name: EndScreen
*   Module number: 1.7.3.1
*   Description: Wraps up the screen display after there has been a full
*               screen displayed.
*   Passed Variables: advance
*   Returns: None
*   Global Variables Used: linecount
*   Global Variables Changed: linecount
*   Files Read: None
*   Files Written: None
*   Modules Called: ClearScreen, Header
*   Calling modules: ShowStats
*
*   Author: Capt Frank W. DeMarco
*   History:
*   1.0 Frank W. DeMarco 10/15/85 - input original code
*****)

```

```

procedure EndScreen(advance: integer);

```

```

var
  i : integer;

```

```

begin (* Procedure EndScreen *)

```

```

  for i := 1 to (advance - 1) do
  begin
    write ('*   |   |   |   |   |   |   |   *');
    writeln ('   |   |   |   *');
  end;
  writeln (ALINEP1,ALINEP2);
  for i := 1 to 26 do
    write (' ');
  write ('Press RETURN to continue.'):
  readln;
  ClearScreen;
  Header;
  linecount := 0

```

```

end; (* Procedure EndScreen *)

```

```

(*****
(* Start of main part of procedure: ShowStats *)
*****)

```

```

begin (* Procedure ShowStats *)

```

```

  while (framenode <> nil) do
  begin

```

```

    lesson_number := (framenode^.ltf_num div 10000);

    frame_number := (framenode^.ltf_num div 1000);
    frame_number := (frame_number * 1000);
    frame_number := (framenode^.ltf_num - frame_number);

    if (framenode^.s_ans = framenode^.c_ans) then
        num_right := num_right + 1.0
    else
        num_wrong := num_wrong + 1.0;
    num_seen := num_seen + 1.0;
    case framenode^.s_ans of
        'A' : num_A := num_A + 1;
        'B' : num_B := num_B + 1;
        'C' : num_C := num_C + 1;
        'D' : num_D := num_D + 1;
        'E' : num_E := num_E + 1;
        'Y' : num_Y := num_Y + 1;
        'N' : num_N := num_N + 1;
    end;
    framenode := framenode^.next
end;

percent_right := ((num_right / num_seen) * 100.0);
percent_wrong := ((num_wrong / num_seen) * 100.0);
tot_r := trunc(num_right);
tot_w := trunc(num_wrong);

if (linecount < 17) or (choice in ['h','H']) then
begin
    write ('* ',lesson_number:1,' ' : ',frame_number:3,' ' : ');
    write (num_A:2,' ' : ',num_B:2,' ' : ',num_C:2,' ' : ',num_D:2,' ' : ');
    write (num_E:2,' ' : ',num_Y:2,' ' : ',num_N:2,' ' * ');
    write (tot_r:2,' ' : ',tot_w:2,' ' : ');
    write (percent_right:5:1,' ',percent_wrong:5:1,' * ');
    writeln;
    linecount := linecount + 1
end
else
begin
    advance := 23 - (linecount + 5);
    EndScreen(advance)
end;

end; (* Procedure ShowStats *)

(*****
(* Start of main part of procedure: Display *)
*****)

begin (* Procedure Display *)

    filenode := filehead^.next;

```

```

    framemode := framehead;

    repeat
        BuildFrameLL;
    until (filenode = nil);

    InitDisplay;
    framemode := framehead^.next;
    ShowStats;

end;    (* Procedure Display *)

(*****
*   Date: 10/15/85
*   Version: 1.0
*
*   Name: FinalScreen
*   Module number: 1.8
*   Description: Wraps up the screen display after all statistics have been
*               processed.
*   Passed Variables: None
*   Returns: None
*   Global Variables Used: advance, linecount, dindx
*   Global Variables Changed: advance, dindx
*   Files Read: None
*   Files Written: None
*   Modules Called: ClearScreen
*   Calling modules: program CAI_Statistics
*
*   Author: Capt Frank W. DeMarco
*   History:
*   1.0 Frank W. DeMarco 10/15/85 - input original code
*****)

procedure FinalScreen;

begin    (* Procedure FinalScreen *)

    advance := 20 - (linecount + 5);
    for dindx := 1 to (advance - 1) do
        begin
            write ('*      |      |      |      |      |      |      |      *');
            writeln ('      |      |      |      *');
        end;
    writeln (ALINEP1,ALINEP2);
    for dindx := 1 to 26 do
        write (' ');
    write ('Press RETURN to end program. ');
    readln;
    ClearScreen;

end;    (* Procedure FinalScreen *)

```

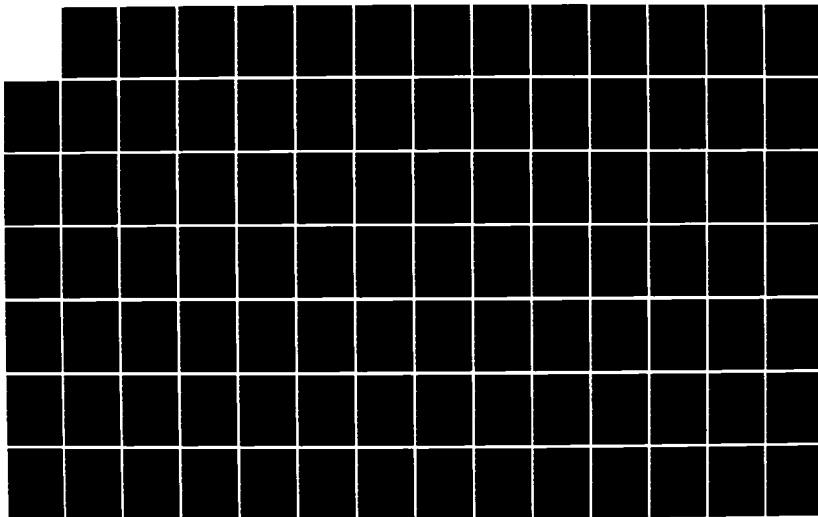
AD-A163 842

COMPUTER ASSISTED INSTRUCTION FOR THE 'C' PROGRAMMING
LANGUAGE ON THE ZEN. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. F W DENARCO
DEC 85 AFIT/GCS/HA/85D-2 F/G 9/2

2/3

UNCLASSIFIED

NL






```

(*****)
(* Start of main driver:  Program CAI_Statistics  *)
(*****)

begin  (* Program CAI_Statistics *)

  ClearScreen;
  QueryUser(choice);
  ClearScreen;
  Init;

  writeln ('One moment please... reading statistical collection file. ');
  while (character in ['1'..V_LESSONS]) and not (eof(infile)) do
    ReadStats;

  Sort(stats);
  Header;

  linecount := 0;
  dindx := 1;
  repeat
    Display;
    dindx := dindx + 1;
  until (stats[dindx] = 0);

  if (choice in ['s','S']) then
    FinalScreen;
  else
    writeln (ALINEP1,ALINEP2)

end.  (* Program CAI_Statistics *)

```

Appendix C

Files Used by Program "CAI"

File "INTRO"

```
#      WW      WW  EEEEEEEE  LL      CCCCCC  000000  MMM  MMM  EEEEEEEE
#      WW WW WW  EE          LL      CC          00    00  MMM  MMM  EE
#      WW WW WW  EEEEE  LL      CC          00    00  MM MM MM  EEEEE
#      WWW  WWW  EE          LL      CC          00    00  MM MM MM  EE
#      WWW  WWW  EEEEEEEE  LLLLLLLL  CCCCCC  000000  MM    MM  EEEEEEEE
```

```
TTTTTTTTTT  00000000
TT          00    00
TT          00    00
TT          00    00
TT          00000000
```

```
""  CCCCCCCCCCCCCCCC  ""
""  CCCCCCCCCCCCCCCC  ""
CCC
CCC
CCC
CCC
CCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCC
```

```
# THE COURSE YOU ARE ABOUT TO TAKE WAS WRITTEN BY CAPT FRANK DEMARCO
# IN PARTIAL FULFILLMENT OF HIS MASTERS DEGREE IN INFORMATION SYSTEMS.
```

```
# THIS COURSE IS DESIGNED AS AN INTRODUCTORY LEVEL COURSE FOR THE "C"
# PROGRAMMING LANGUAGE. THE OBJECTIVE OF THE COURSE IS TO PROVIDE
# ENOUGH INFORMATION TO THE STUDENT SO THAT IT MAY BE POSSIBLE FOR
# THE STUDENT TO BEGIN USING THE "C" LANGUAGE FOR HIS/HER PROGRAMMING
# NEEDS.
```

```
# THE COURSE, AS IT CURRENTLY EXISTS, CONSISTS OF SIX LESSONS.
```

File "MENU"

```
*****
*
*           SELECT THE LESSON YOU WISH TO TAKE FROM THE FOLLOWING:
*
*****
*   STATUS   LESSON #   LESSON TITLE
*   -----   -
* @           1       GETTING STARTED WITH C
*
* @           2       VARIABLES, CONSTANTS, OPERATORS, EXPRESSIONS
*
* @           3       PROGRAM CONTROL STATEMENTS
*
* @           4       POINTERS AND ARRAYS
*
* @           5       STRUCTURES
*
* @           6       INPUT AND OUTPUT
*
*****
*   NOTE: A "STATUS" OF "+" INDICATES LESSON SUCCESSFULLY COMPLETED.
*
*****
```

File "LESSON1"

```
#      WW      WW  EEEEEEEE  LL          CCCCCC  000000  MMM  MMM  EEEEEEEE
#      WW WW WW  EE          LL          CC          00   00  MMM  MMM  EE
#      WW WW WW  EEEEE  LL          CC          00   00  MM MM MM  EEEEE
#      WWW  WWW  EE          LL          CC          00   00  MM MM MM  EE
#      WWW  WWW  EEEEEEEE  LLLLLLLL  CCCCCC  000000  MM    MM  EEEEEEEE
```

```
#
#
#      TTTTTTTTTT  00000000
#      TT          00      00
#      TT          00      00
#      TT          00      00
#      TT          00000000
```

```
#      LL          EEEEEEEE  SSSSSS  SSSSSS  000000  NN    NN    111
#      LL          EEEEEEEE  SSS SSS  SSS SSS  00000000  NNN  NN    1 11
#      LL          EE          SSS      SSS      00      00  NNNN  NN    11
#      LL          EEEEE  SSSS      SSSS      00      00  NN NN NN    11
#      LL          EEEEE  SSSS      SSSS      00      00  NN NN NN    11
#      LL          EE          SSS      SSS      00      00  NN NNNN    11
#      LLLLLLLL  EEEEEEEE  SSS SSS  SSS SSS  00000000  NN    NNN  11111111
#      LLLLLLLL  EEEEEEEE  SSSSSS  SSSSSS  000000  NN    NN    11111111
```

```
# THE LESSON YOU ARE ABOUT TO TAKE CONTAINS INTRODUCTORY INFORMATION
# ON THE COURSE AND SOME GENERAL INFORMATION ON C PROGRAMMING.
```

```
# THE LESSON CURRENTLY CONSISTS OF FIVE TOPICS.
```

```
# The Lesson Breakdown Is As Follows:
```

```
# Topic 1: Introduction to C CAI course - This topic gives a short
# introduction to the overall course structure and some of
# the particulars used in the course. (Approx. time = 5 min.)
```

```
# Topic 2: C Program Organization - This topic discusses the overall
# organization and structure of a typical C program.
# (Approx. time = 15 min.)
```

```
# Topic 3: C Program Environment - This topic gives a description of
# the overall C programming environment covering such items
# as "compiling", and "linking". (Approx. time = 10 min.)
```

```
# Lesson Breakdown Continued:
```

```
# Topic 4: Your First C Program - This topic states a problem to be solved
# and presents a solution for you to help familiarize you with
# C program statements. (Approx. time = 10 min.)
```


12 these six areas are further broken into small topic areas. The
12 goal in organizing the course in this way is to make it easier
12 to understand as well as speed up the process of subject review.

12 The following is a lesson breakdown of the course:

12 LESSON #	12 LESSON TITLE
12 -----	12 -----
12 1	12 Getting Started With C
12 2	12 Variables, Constants, Operators, Expressions
12 3	12 Program Control Statements
12 4	12 Pointers and Arrays
12 5	12 Structures
12 6	12 Input and Output

13B:110

11Frame 110 QM

12Let's see if you have been paying attention. How many lessons did I say are
12in this course?

13A Four

13

13B Five

13

13C+ Six

13

13D Seven

14 Very good! You are paying attention.

14 B:115

15ABD No. The correct answer is Six lessons ("C").

15 B:115

15E I'm sorry, "E" was not one of your choices.

15 B:110

11Frame 115 T

12 What you just saw was an example of one of three types of questions I
12 can ask during the presentation of this course. The other types are
12 True/False and Yes/No questions. The responses that I can recognize
12 are as follows:

12 Question Type	12 Valid Responses
12 -----	12 -----
12 Multiple Choice	12 A, B, C, D, E
12 True or False	12 True, False, T, F
12 Yes or No	12 Yes, No, Y, N

12 Note: For True/False or Yes/No questions I will only look at the
12 first letter of your response, so to save time it's best to
12 enter only T, F, Y, or N. (Answers may be in lower case.)

13B:120

11Frame 120 QP

12Let's give it a try.

12

12This is an example of a True/False type question. (True or False)

13Y

14 You are absolutely correct.

14 B:125

15 Wrong! Are you yanking my electrons?

15 B:125

11Frame 125 T

12 As you will no doubt notice, there is a test at the end of each
12 lesson. In order for you to receive credit for taking this course
12 you must successfully pass each of these tests.

12

12 There is no set lesson order in taking this course, nor is there a
12 requirement to view each topic before taking a lesson test. It is
12 suggested that you do take the course in the order established for
12 reasons of material continuity and in order to enhance understanding.

12

12 It is very important that you do not interrupt the CAI program once
12 it has been started. Your progress is only recorded at the end of
12 each lesson topic. Please exit the program by answering "X" at the
12 topic selection menu and the lesson selection menu.

12

12 You can check your lesson and topic progress at the selection menus
12 by observing the "status" column displayed.

13B:130

11Frame 130 T

12 While taking this course you can be an invaluable aid in making it
12 better by taking note of errors in the course material. If you
12 should notice an error or believe something to be in error, just
12 make a note of where the error appears.

12

12 To make this task easier, I display for you all the necessary infor-
12 mation. Just record the lesson number, topic number, and of course
12 the frame number of the frame where the error appears. Recording
12 only the frame number will be of little help since each lesson could
12 have a frame with the same number.

12

12 I just have one more thing to mention to you before I return you to
12 the topic selection menu.

13B:135

11Frame 135 T

12 A word about the lesson tests.

12

12 The last lesson topic for each lesson is a test over the material
12 covered in the lesson. As I mentioned before, these tests must be
12 passed in order to receive credit for taking this course.

12

12 When you take a test, you will be given information during the test
12 which will help you in locating the material that gave you problems
12 on the test. The way in which this is done is by reference to the

12 lesson topic and frame number where the material was covered. An
 12 example of two types of feedback you might see are:

12 Right. (2,245) <OR> Wrong. (2,245)

12 The reference follows the format of: (lesson topic #,topic frame #)

12 *** This concludes this topic area. ***

13END

21Frame 300 T C PROGRAM ORGANIZATION

22 This topic will discuss the overall organization of a typical
 22 C program. For ease of understanding, I will restrict the
 22 discussion to a program that is contained in one source file.

22

22 The organization of the program file would look like the following:

22

22 1. Preprocessor Statements Section

22

22 2. Global Variable Declarations Section

22

22 3. Function(s)

22

22 4. Main Program Driver

22

22 Each of the above will be discussed in this topic section.

23B:305

21Frame 305 T

22 *** Preprocessor Statement Section ***

22

22 Through the use of a preprocessor, the C compiler has the capability
 22 of: file inclusion, token substitution, and conditional compilation.

22

22 Preprocessor statement lines are defined in the C program by the use
 22 of a # as the first character on a line. These lines may appear
 22 anywhere in the program, but it is a good programming practice to
 22 place them at the beginning.

22

22 We'll take a quick look at each of these preprocessor capabilities.

23B:310

21Frame 310 QM

22Which one of the following is "not" a capability of the C preprocessor?

23A Conditional Compilation

23

23B Token Substitution

23

23C+ Function Definition

23

23D File Inclusion

24 Very good, you're so right.

24 B:315
25ABD No. Answer "C" is the correct answer.
25 B:315
25E I'm sorry, "E" was not one of your choices.
25 B:310
21Frame 315 T
22 * File Inclusion *
22
22 The preprocessor control line of the form: #include "filename"
22 will include the contents of the file specified in the source
22 program file. (Note: The shown quotation marks are needed.)
22
22 In addition, a control line of the form: #include <filename>
22 will include the contents of the "system" file specified.
22
22 For example: #include <stdio.h> is the usual statement for
22 including the file that contains the standard I/O functions
22 for use with C. We will see more of this later.
22
22 One more thing: An included file may also have files included.
22
22 This of course should be done cautiously to avoid confusion.
23B:320
21Frame 320 QP
22#include file.dat is a valid C preprocessor "file inclusion" statement.
22(True or False)
23N
24 That's right. You need to have quotation marks around the file name.
24 B:325
25 Wrong. Quotation marks are needed around the file name.
25 The correct form of the statement is #include "file.dat" .
25 B:325
21Frame 325 T
22 * Token Substitution *
22
22 The preprocessor control line of the form:
22
22 #define token-name token-replacement
22
22 will substitute the value of the token-replacement for each occurrence
22 of the token-name throughout the program.
22
22 For example: If you have a value that might change with time, such
22 as a mortgage rate, you could use the #define to make future
22 program changes easier like this ==> #define interest .11
22
22 It is easy to see that this capability can be a real time saver. Not
22 only will it make future program changes easier but it will, with wise
22 token-name choices, produce an easier to read and maintain program.
22 Note: #undef token-name is used to cancel the token-replacement.
23B:330

21Frame 330 QM

22Which of the following is a valid C preprocessor "token substitution"
22statement?

23A #define pay_grade = 11

23

23B #declare pay_grade 11

23

23C+ #define pay_grade 11

23

23D #declare pay_grade = 11

24 Correct. Keep up the good work.

24 B:335

25A No. There is no "=" in the valid form of the statement. Answer "C" is
25 the correct response.

25 B:335

25BD No. I think you missed something. Let's look at that again.

25 B:325

25E I'm sorry, "E" was not one of your choices.

25 B:330

21Frame 335 T

22 * Conditional Compilation *

22

22 You can cause the compiler to skip sections of your source code by
22 using the conditional preprocessor control statements of:

22 #if, #ifdef, #ifndef, #else, and #endif.

22

22 The statement #if constant-expression will evaluate to "true" if
22 the constant-expression is a non-zero value.

22

22 The statement #ifdef identifier will evaluate to "true" if the
22 identifier had previously been defined using the #define.

22

22 The statement #ifndef identifier will evaluate to "true" if the
22 identifier had not been previously defined using the #define.

22

22 Following the above statements would be statements that you would
22 want to be executed based on the outcome of the statement test.

23B:340

21Frame 340 T

22 * Conditional Compilation Continued *

22

22 The statement #else would be used to identify an alternate section
22 of code to be executed if the outcome of the #if.. statement test
22 evaluates to false.

22

22 The statement #endif is used to terminate an #if.. #else structure.

22

22 Example: #ifdef employed /* check to see if "employed" #define(d) */
22 {
22 executable statements;
22 }
22 #else /* else "employed" is not #define(d) */

```

22      (
22      alternate executable statements;
22      )
22      #endif
23B:345
21Frame 345 QP
22#ifdef is an example of a C preprocessor "conditional compilation"
22statement. (True or False)
23Y
24 Yes, that's right.
24 B:350
25 Sorry, the answer is "True".
25 B:350
21Frame 350 T
22 *** Global Variable Declarations Section ***
22
22 Whenever a variable is declared independent of a function, it is
22 called a "global" variable. The "scope of a variable" refers to
22 the area where a declared variable is recognized. If you intend
22 to use the same variable in different portions of your program,
22 then it may be desirable to declare the variable as being global.
22
22 When you declare a global variable, remember that its "scope" is
22 only those functions (procedures) that physically follow it in the
22 program. (Note: An exception to this involves the "extern" decla-
22 ration statement which I will cover later.)
22
22 Let's look at an example...
23B:355
21Frame 355 T
22 * Global Variable Declaration Example *
22
22 #include <stdio.h>
22 int sum; /* Global Variable "sum" */
22 main() {
22     sum = 100;
22     add();
22     add();
22     printf("%d",sum);
22 }
22 add() {
22     sum = sum + 100;
22 }
22
22 This program would print out the value 300.
22
22 Don't worry if you don't follow everything in this example. You will.
23B:360
21Frame 360 QM
22The "scope" of a global variable refers to ...
23A the number of variables affected by the global variable.
23

```

23B the extent the global variable is used in the program.

23

23C the area preceding the global variable declaration.

23

23D+ the area where a declared global variable is recognized.

23

23E the mouthwash of the global variable.

24 Right.

24 B:365

25ABC No. The correct answer is "D".

25 B:365

25E I'm sorry, variables don't have mouths.

25 B:360

21Frame 365 T

22 *** Functions ***

22

22 Following global variable declarations in our typical program example
22 is the area where we define our functions. The structure of a function
22 looks like this:

```
22 return-type function-name(arguments, if any)
22     argument declarations, if any
22 {
22     declarations
22     executable statements
22     return statement, if any
22 }
```

22

22 A function has certain required parts. Here's an example of a function
22 that fills the requirement:

```
22                                     function1() {}
```

23B:370

21Frame 370 T

22 * Functions Continued *

22

22 The previous example of a function is an example of a dummy function.
22 The function doesn't actually do anything, but does qualify as a
22 function.

22

22 Let's look at each of the parts of the function structure.

22

22 The "return-type" in front of the function name identifies the type
22 of result the function will return to the function that called it.
22 If this return-type is not explicitly named, then it defaults to an
22 integer type. If a function returns a value other than an integer
22 and it is physically located after the calling function then it is
22 also necessary to declare the "function" as that return type in the
22 calling function.

23B:375

21Frame 375 T

22 * Functions Continued *

22

22 For example: float numval, function1
22
22 The above statement would be in the declaration section of the
22 function that calls function1.
22
22 The function declaration: float function1(numval)
22
22 would be used to identify the called function "function1" and state
22 that the value to be returned by this function is of type float.
22 Note: We will discuss "float" later.
22
22 That brings us to the "function-name" part of the function structure.
22
22 The function-name can be of any length but must start with a letter.
22 Note: The character _ (underscore) is considered a letter in C.
23B:380
21Frame 380 T
22 * Functions Continued *
22
22 The function-name can consist of any combination of letters and digits
22 as long as it starts with a letter and does not spell a C keyword.
22 Note: Keywords will be discussed later.
22
22 Following the function-name is a required set of parentheses ().
22 Inside the parentheses is where the list of passed arguments goes.
22 Each argument is separated by a comma and appears in the order in
22 which the calling function lists them in its calling statement.
22
22 Next in our function structure is the area for argument declaration.
22 This is the area where we identify the "types" of the passed arguments.
22
22 For example: float function1(x,y)
22 float x, n; <== argument declaration
23B:385
21Frame 385 QM
22Which of the following is required to follow the function-name in a function
22declaration statement?
23A " "
23
23B+ ()
23
23C / /
23
23D # #
24 Very good.
24 B:390
25ACD No. I think your falling asleep. Let's take a step back.
25 B:380
25E I'm sorry, "E" was not one of your choices.
25 B:385
21Frame 390 T
22 * Functions Continued *

22
 22 If an argument is not explicitly declared it defaults to type integer.
 22
 22 Following the argument declarations is a required set of braces {}.
 22 Inside the braces is where the function's declarations, executable
 22 statements, and return statement goes. Each statement in this area,
 22 as in the argument declaration area, is terminated by the use of a
 22 semi-colon.
 22
 22 We'll look at each of the three areas between the braces.
 23B:395
 21Frame 395 QP
 22If an argument passed in to a function is not explicitly declared its
 22"type" defaults to an integer. (True or False)
 23Y
 24 You are absolutely correct.
 24 B:400
 25 No. That statement is correct.
 25 B:400
 21Frame 400 T
 22 * Functions Continued *
 22
 22 Here are the three areas between the {} braces.
 22
 22 1. A function will usually have a need to have local parameters and
 22 variables defined in order to do its job in the program. The
 22 function's declaration section within the braces is where these
 22 declarations take place.
 22
 22 2. Following the local declarations is the function's executable state-
 22 ments. These are the statements to be executed by the function prior
 22 to returning control to the calling function.
 22
 22 3. The "return" statement is where you identify the variable that is
 22 to be passed back to the calling function. The "return" statement
 22 can be a bit confusing. There are three forms in which the statement
 22 can appear.
 23B:405
 21Frame 405 T
 22 * Functions Continued *
 22
 22 The most common form of "return" is: return(expression);
 22
 22 The "expression" can be any valid expression, such as value * 2
 22 or just value. In either case the final value will be passed
 22 back to the calling function as the value of the function-name.
 22 Remember, it is an integer unless explicitly declared otherwise.
 22
 22 Another form of the "return" is: return expression;
 22
 22 The elimination of the parentheses also eliminates the confusion
 22 of whether or not "return" is a function (which it isn't).

23B:410

21Frame 410 T

22 * Functions Continued *

22

22 The last valid form of "return" is: return;

22

22 This case has the same effect as leaving out the return statement.

22

22 In either case, no value is returned to the calling function and only global variables used by the function would be changed as a result of the called function being executed.

22

22 Important Note: ** Do Not Use ==> return(); as this will cause a compile error since "return" is not a function.

22

22

22 We only have one short area left to cover. But first a question.

23B:415

21Frame 415 QM

22 Which of the following is "not" a valid return statement?

23A return(expression);

23

23B return expression;

23

23C+ return();

23

23D return;

24 Very good!

24 B:420

25ABD Wrong. That is a valid return statement. "C" is the invalid one.

25 B:420

25E "E" was not a given choice. Please try again.

25 B:415

21Frame 420 T

22 *** Main Program Driver ***

22

22 This area of the program is usually located at the end of the source program file. It is the required function that starts and ends the programs execution. There must be a function by the name of main() somewhere in your C program.

22

22 The organization of the function "main()" is the same as for the functions we just covered. I bet that makes you happy!

22

22

22 Well that about does it for this topic. Let's take one more look at the overall construction of a typical C program before returning to the topic menu.

23B:425

21Frame 425 T

22 *** Review of C Program Organization ***

22

22 The organization of the program file would look like the following:

22
22

22 1. Preprocessor Statements Section

22

22 2. Global Variable Declarations Section

22

22 3. Function(s)

22

22 4. Main Program Driver

22

22

22 *** This concludes this topic area. ***

23END

31Frame 500 T C PROGRAM ENVIRONMENT

32 This topic will discuss the overall C programming environment.

32

32 We will follow the complete process of creating a C program from the
32 writing of code to the execution of the resultant executable program.

32

32 Although this may sound like a lot to cover, it really isn't.

32

32 To get us started let's take a look at the process as a whole.

32

32 The following is an outline of the steps we will cover:

32

32 1. Create Source File

32

32 2. Compile Source File

32

32 3. Error Correction

32

32 4. Link Object Code Files

32

32 5. Run Executable Code File

32

32

32

32

32

32

32

32

32

32

32

32

32

32

32

32

32

32

32

32

32

32

32

32

32

32

32

32

32

32 Let's get started ...

33B:505

31Frame 505 T

32 *** Create Source File ***

32

32

32

32

32

32

32

32

32

32

32

32

32

32

32

32

32

32

32

32

32

32

32

32

32

32

32

32

32 The most important aspect of computer programming in any language is
32 the ability to put your thoughts into computer code. Many experienced
32 programmers feel that the best way to write clear, concise, effective
32 code is to write in plain english "what" it is that needs to be done.

32 Once the "what" has been identified you can start working on the "how"
32 do I do it question. This brings us to a controversial topic, that of
32 where can I do my best program development? Do I do it on paper, or
32 do I sit at a computer terminal and "create" as I go. Well it all
32 depends on who you talk to as to which way is better, but the person
32 who wrote the program your using now prefers to "create" his programs
32 at the computer terminal. Of course it is not always up to you where
32 you do you programming. Computer time costs money after all, and you
32 and/or your boss should be concerned about such factors.

33B:510

31Frame 510 T

32 * Create Source File Continued *

32

32 Whichever way you finally decide to do it, you are going to need a
32 way to put the code you have written into a source file for use on
32 the computer. This calls for the use of a text editor. The more
32 familiar you are with the text editor the easier and faster you can
32 input your code into a source file. Remember, chances are you will
32 have to make error corrections or update your program at some point.

32

32 So, learn your text editor and use it often.

32

32 Once you have created your C program source file using a text editor,
32 it is time to compile it.

33B:515

31Frame 515 T

32 *** Compile Source File ***

32

32 At this point in our C program development we have one source file.
32 The next step is to translate the "source code" in the source file
32 into "object code" in an object file. This translation is accom-
32 plished by the C compiler.

32

32 The C compiler is actually a program that performs three basic
32 functions using three distinct programs.

32

32 1. The C Preprocessor

32

32 2. The C Compiler

32

32 3. The C Assembler

33B:520

31Frame 520 QP

32The C compiler is actually three programs in one. (True or False).

33Y

34 That's right. A preprocessor, compiler, and assembler all in one.

34 B:525

35 No. Are you sure you read that last frame? Let's see it again.

35 B:515

31Frame 525 T

32 * Compile Source File Continued *

32

32 First, the C "preprocessor" scans the source code for preprocessor
32 statements (# statements) and performs all indicated actions.

32

32 Second, the C "compiler" translates the C language statements into
32 computer assembly language statements.

32

32 Last, the C "assembler" translates the assembly language statements
32 into the object code and places it in an object file.

32

32 This last step occurs IF you have not made any C syntax errors!

33B:530

31Frame 530 T

32 *** Error Corection ***

32

32 When you compile your C program, it is possible that you may have
32 made one or two syntax errors. Don't feel bad, it can happen to
32 even the best programmer (once in a while). If this unfortunate
32 occurrence takes place, you can rest assured that the C compiler
32 will let you know.

32

32 The C compiler will report any syntax errors that it encounters
32 while compiling your source code. In order to achieve the goal
32 of syntax error free object code, it may be necessary to go through
32 several iterations of "compile & correct".

32

32 This process will require changes to your source file, which is
32 a reason why you should know how to use your text editor program
32 inside and out.

33B:535

31Frame 535 T

32 *** Link Object Code Files ***

32

32 Once you have successfully produced an "object code" file, it is time
32 to move on to creating an executable program file.

32

32 The C "linker" is a program that is used to link together object files
32 into an executable "machine code" file. The C linker will take all
32 specified object files as well as any needed C library functions
32 and create for you one executable program file.

32

32 This feature allows for the creation of user functions that can be
32 used in a variety of programs by merely linking them into the new
32 program. These functions can then be called by the program when
32 needed. This will save you many hours of redundant work.

32

32 What now? You ask. Well you'll see, but first a question.

33B:540

31Frame 540 QM

32Which one of the following programs will create an executable program file
32from one or more object files?

33A Compiler

33

33B Chainer

33

33C+ Linker

33

33D Preprocessor

33

33E Assembler

34 Correct.

34 B:545

35ABCE Wrong. The "Linker" creates the "executable" machine code program file.

35 B:545

31Frame 545 T

32 *** Run Executable Code File ***

32

32 Now that you have the executable program file you can sit back and
32 start the seemingly long process of "logic" testing your program.

32

32 That's right! It's run time!

32

32 At this point, all syntax errors have been corrected and you have
32 successfully created an executable program file. Now you can test
32 your program to your heart's content and make all those changes
32 and/or enhancements to your pride-and-joy (your C program).

32

32 Before I return you to the topic selection menu, I would like to

32 give you a picture of the process described in this topic area.

32 I made no mention of "how" to do the steps only "what" steps needed

32 to be done. The commands to "compile" and "link" differ from system

32 to system, but are similar enough to show an example.

33B:550

31Frame 550 T

32 *** C Environment Example ***

32

32 The following is an example of the steps needed to create and run
32 an executable program file.

32

32

32 1. Create Source File =====> progname.c (using a text editor)

32

32 2. Compile Source File =====> cc progname.c (using C Compiler)

32

32 3. Error Correction =====> (As needed) (using a text editor)

32

32 4. Link Object Code Files =====> clink progname (using C Linker)

32

32 5. Run Executable Code File ==> progname (type program name)

32

32 6. Refine Program Execution ==> (As needed) (using a text editor)

33END

41Frame 700 T YOUR FIRST C PROGRAM

42 This topic will develop an actual working C program for you to examine.

42

42 In order to provide a problem solving structure, here is an outline
42 of the steps that I will be discussing:

42

42 1. Problem Definition

42

42 2. English Language Problem Solution

42

42 3. English Language - C Language Translation

42

42 4. C Language Problem Solution

43B:705

41Frame 705 T

42 *** Problem Definition ***

42

42 The first thing that needs to be done is to define the problem.

42

42

42 The program I want to develop is one that will:

42

42 1. Take a one line input from the keyboard, and

42

42 2. Display the input line "centered" on the terminal screen.

42

42

42 I will show one way to accomplish this and then allow you to choose

42 whether you want to view an alternate solution.

43B:710

41Frame 710 T

42 *** English Language Problem Solution ***

42

42 After thinking out the problem, I arrived at the following five step
42 solution:

42

42 1. Define and initialize the storage area for one line of input.

42

42 2. Prompt user for one line of input.

42

42 3. Read in the one line of input keeping track of number of
42 characters read.

42

42 4. Calculate number of spaces to precede line for "centered" output.

42

42 5. Print out the input line "centered" on screen.

43B:715

41Frame 715 T

42 *** English Language - C Language Translation ***

42

42 Changing the english problem solution into C language statements we get:

42

42 1. #define CHARIN 80

42 char input_line[CHARIN];

42 for (i=0; i < CHARIN; i++)

42 input_line[i] = ' ';

42 2. printf("\nPlease enter one (1) line of text to be centered.\n");

42 3. while ((c = getchar()) != '\n') {

42 input_line[i] = c;

42 i++; }

42 4. advance = (40 - (i / 2));

42 5. for (ival=0; ival < advance; ival++)

42 putchar(' ');

42 printf("%s",input_line);

43B:720

41Frame 720 T

42 *** C Language Problem Solution ***

42

42 Now that we have the C statements needed for problem solution, all
42 we need to do is declare the variables we used and put the code
42 into a function called "main".

42

42

42 The following topic frame gives one complete solution.

43B:725

41Frame 725 T

42 #define CHARIN 80

42

42 main() {

42 char c, input_line[CHARIN];

42 int i, ival, advance;

42

42 for (i=0; i < CHARIN; i++)

42 input_line = ' ';

42 printf("\nPlease enter one (1) line of text to be centered.\n");

42 i = 0;

42 while ((c = getchar()) != '\n') {

42 input_line[i] = c;

42 i++; }

42 advance = (40 - (i / 2));

42 for (ival=0; ival < advance; ival++)

42 putchar(' ');

42 printf("%s",input_line); }

43B:730

41Frame 730 T

42 At this stage of the course, I don't feel that I should take the time
42 to explain each C statement used in this example program. Please rest
42 assured that I plan to explain all the statements used here as well as
42 a multitude of others later in this course.

42

42 The example solution is by no means the only solution to the stated
42 problem, it is only one of many "correct" solutions. It also is not
42 a fool proof solution (Input of > 80 characters is not checked for.).

42

42

42 The rest of this topic area contains an alternate solution to this
42 same problem. Other C statements are shown, but again no explanation
42 is given.

43B:735

41Frame 735 QP

42Do you wish to see another example solution? (Yes or No)

43Y

44 Great! Let's take a look at one.

44 B:740

45 Alright. I will honor your decision.

45 B:780

41Frame 740 T

42 *** Alternate Solution ***

42

42 In the solution we just finished with, the whole solution was con-
42 tained in the "main" function. This practice is not a good one to
42 get in the habit of. A better way to solve programming problems
42 is to break the problem solution into small "modules" or, in the
42 case of C, functions.

42

42 Earlier I identified several steps to be accomplished in order to
42 solve the example problem. Each main step could be done by its own
42 separate function or we could combine two or more steps into one
42 function. Let's see what we end up with if we use this latter
42 approach.

43B:745

41Frame 745 T

42 * Alternate Solution Continued *

42

42 The first step was "Define and initialize the storage area for one
42 line of input."

42

42 This of course can be accomplished using a global variable called
42 "CHARIN" having value 80, an array declaration in function "main"
42 of the form: char input_line[CHARIN], and a statement that "blanks"
42 out the array using a "for" statement for control.

42

42 These three statements look like the following:

42

42 (1) (2) (3)

42

42 #define CHARIN 80 ; char input_line[CHARIN]; ; for (i=0; i < 80; i++)
42 input_line[i] = ' ';

43B:750

41Frame 750 T

42 The second step of "Prompt user for one line of input" and the third
42 step of "Read in the one line of input keeping track of number of
42 characters read" can be combined into one.

42

42 For this we can define a function called "task1" that would look like:

42

42 task1(input_line, i)

42 char input_line[];

42 int i,

42 { char c;

42

42 printf("\nPlease enter one (1) line of text to be centered.\n");

42 while ((c = getchar()) != '\n')

42 { input_line[i] = c;

42 i++; }

42 return(i); }

43B:755

41Frame 755 T

```
42 The fourth step of "Calculate number of spaces to precede line for
42 'centered' output" and the fifth step of "Print out the input line
42 'centered' on screen" can be combined into one.
42
42 For this we can define a function called "task2" that would look like:
42
42 task2(input_line, i)
42     char input_line[];
42     int i;
42 { int advance, ival;
42
42     advance = (40 - (i / 2));
42     for (ival=0; ival < advance; ival++)
42         putchar(' ');
42     printf("%s",input_line); }
```

43B:760

41Frame 760 T

```
42 That just leaves one function to write. That being function "main".
42
42 I will show two different "main" functions that do the same thing.
42
42 main()                                | main()
42 {                                     | {
42     char input_line[CHARIN];         |     char input_line[CHARIN];
42     int i;                           |     int i;
42                                     |
42     for (i=0; i < CHARIN; i++)        |     for (i=0; i < CHARIN; i++)
42         input_line[i] = ' ';         |         input_line[i] = ' ';
42     i = 0;                           |     i = 0;
42     i = task1(input_line, i);         |     task2(input_line, task1(input_line, i));
42     task2(input_line, i);             | }
42 }                                     | }
```

43B:765

41Frame 765 T

```
42 *** A Variation To The Problem ***
42
42 Now that we have solved the example problem two different ways, how
42 about making a slight improvement to it. What if we wanted to clear
42 the screen before we displayed the "centered" line on the screen?
42
42 Well, this can be done fairly easily with the following function:
42
42 task3()
42 {
42     putchar('\033');
42     putchar('H');
42     putchar('\033');
42     putchar('J'); }
42
42 Now all we need to do is call "task3" from "task2" before the
42 loop that does the "putchar(' ')".
```

43B:770

41Frame 770 T

42 This is what the "task2" function would then look like:

```
42
42 task2(input_line, i)
42     char input_line[];
42     int i;
42     { int advance, ival;
42
42         advance = (40 - (i / 2));
42         task3();
42         for (ival=0; ival < advance; ival++)
42             putchar(' ');
42         printf("%s",input_line); }
42
```

42 Of course, "task3" would be located ahead of "task2" in the program
42 source file.

43B:775

41Frame 775 T

42 Let's see what our program source file would look like if we use
42 this alternate program solution with the "clear screen" function.

```
42
42         #define CHARIN 80
42
42         +--> task3()
42         |
42 +-----> +--- task2(input_line, i)
42 |
42 | +-----> task1(input_line, i)
42 | |
42 +----- main
42
```

42 The way to read this is: "main" calls "task1", then "main" calls
42 "task2", then "task2" calls "task3".

43B:780

41Frame 780 T

42 *** Lesson One Summary ***

42 Well, that about does it for lesson number one. If you have seen the
42 four subject topics in this lesson, you should now be ready to take
42 the final test. If you feel that you don't understand something well
42 enough to pass the test, please retake the topic that is giving you
42 problems.

42 Topic 1 gave an introduction to the C CAI course structure.

42 Topic 2 gave a description of the C program organization.

42 Topic 3 gave a description of the C program environment.

42 Topic 4 presented a programming example for your inspection.

42
 42 Good Luck on the test.
 43END
 51Frame 900 TT TEST OVER LESSON 1
 52 Welcome to the final test of lesson one. This test consists of ten
 52 questions over material presented in the previous four topic areas.
 52
 52 In order to successfully complete this lesson, you must achieve a
 52 minimum score of 70% (seven out of ten questions correct).
 52
 52 If you miss a question, the correct answer will not be shown. It is
 52 up to you to research the correct answer.
 52
 52 Well, enough said. Let's get on with it. Good luck!
 53B:905
 51Frame 905 QM
 521. After answering a test question in this course, a reference is shown
 52to you so that you can find the place in the lesson where the question
 52originated. The reference is in the format of (#,@) where ...
 53A # = lesson number and @ = frame number
 53
 53B+ # = lesson topic number and @ = topic frame number
 53
 53C # = lesson topic number and @ = lesson line number
 53
 53D # = lesson number and @ = lesson topic number
 54 Right. (1,135)
 54 B:910
 55ACD Wrong. (1,135)
 55 B:910
 55E "E" was not one of your choices.
 55 B:905
 51Frame 910 QP
 522. The three capabilities of the C preprocessor are: file inclusion,
 52token substitution, and conditional compilation. (True or False)
 53Y
 54 Right. (2,305)
 54 B:915
 55 Wrong. (2,305)
 55 B:915
 51Frame 915 QM
 523. Which of the following is a valid C preprocessor "token substitution"
 52statement?
 53A+ #define interest .09
 53
 53B #declare interest .09
 53
 53C #define interest = .09
 53
 53D #declare interest = .09
 54 Right. (2,315)
 54 B:920

55BCD Wrong. (2,315)

55 B:920

55E "E" was not one of your choices.

55 B:915

51Frame 920 QM

524. Which of the following is "not" a valid C preprocessor "conditional compilation" statement?

53A #if

53

53B #ifdef

53

53C #else

53

53D+ #for

53

53E #ifndef

54 Right. (2,320)

54 B:925

55ABCE Wrong. (2,320)

55 B:925

51Frame 925 QP

525. The function name in a function declaration can consist of any combination of letter, digits, or characters on the keyboard.

52(True or False)

53N

54 Right. (2,355)

54 B:930

55 Wrong. (2,355)

55 B:930

51Frame 930 QM

526. Which of the following is the required function in a C program that usually starts and ends execution of the program?

53A start()

53

53B begin()

53

53C+ main()

53

53D driver()

54 Right. (2,320)

54 B:935

55ABD Wrong. (2,320)

55 B:935

55E "E" was not one of your choices.

55 B:930

51Frame 935 QM

527. Which of the following is a list of the three programs contained in the C compiler?

53A Preprocessor, Compiler, Linker

53

53B+ Preprocessor, Compiler, and Assembler

53

53C Compiler, Assembler, and Linker

53

53D Editor, Preprocessor, Assembler

54 Right. (3,515)

54 B:940

55ACD Wrong. (3,515)

55 B:940

55E "E" was not one of your choices.

55 B:935

51Frame 940 QP

528. The C "Linker" is a program that is used to link together one or more
52object files into an executable "machine code" file. (True or False)

53Y

54 Right. (3,530)

54 B:945

55 Wrong. (3,530)

55 B:945

51Frame 945 QM

529. Which of the following is a "Compiler" program that translates the
52C language statements into assembly language statements?

53A Editor

53

53B Preprocessor

53

53C+ Compiler

53

53D Assembler

53

53E Linker

54 Right. (3,520)

54 B:950

55ABDE Wrong. (3,520)

55 B:950

51Frame 950 QM

5210. What is the first thing that needs to be done when solving a computer
52programming problem?

53A+ Define the problem to be solved.

53

53B Write the "english" language solution.

53

53C Do an "english" to "C" language translation.

53

53D Write the "C" language solution.

54 Right. (4,700)

54 B:955

55BCD Wrong. (4,700)

55 B:955

55E "E" was not one of your choices.

55 B:950

51Frame 955 T

52 *** End of Lesson Material ***

52

52 This marks the end of lesson number one. I hope that it was of some
52 benefit to you. I am looking forward to seeing you in lesson number
52 two. I hope that you didn't have too much trouble with the material
52 presented in this lesson. If you did, please voice your comments to
52 your training monitor who will in turn contact the CAI Plans Branch
52 at Keesler AFB, MS.

52

52 Well, let's take a look at how you did with the test ...

53END

File "LESSON2"

```
#   WW   WW  EEEEEEEE LL       CCCCCC  000000  MMM  MMM  EEEEEEEE
#   WW WW WW  EE       LL       CC       00   00  MMM  MMM  EE
#   WW WW WW  EEEEE   LL       CC       00   00  MM MM MM  EEEEE
#   WWW  WWW  EE       LL       CC       00   00  MM MM MM  EE
#   WWW  WWW  EEEEEEEE LLLLLLLL CCCCCC  000000  MM   MM  EEEEEEEE
```

```
#
#
#           TTTTTTTTTT  00000000
#           TT       00   00
#           TT       00   00
#           TT       00   00
#           TT       00000000
```

```
#   LL       EEEEEEEE  SSSSSS  SSSSSS  000000  NN   NN   22222
#   LL       EEEEEEEE  SSS SSS  SSS SSS  00000000 NNN  NN   2222222
#   LL       EE       SSS       SSS       00   00  NNNN NN   2  22
#   LL       EEEEE   SSSS       SSSS       00   00  NN NN NN   222
#   LL       EEEEE       SSSS       SSSS       00   00  NN NN NN   222
#   LL       EE       SSS       SSS       00   00  NN  NNNN   22
#   LLLLLLLL EEEEEEEE  SSS SSS  SSS SSS  00000000 NN   NNN  22222222
#   LLLLLLLL EEEEEEEE  SSSSSS  SSSSSS  000000  NN   NN   22222222
```

```
#
# THE LESSON YOU ARE ABOUT TO TAKE CONTAINS INFORMATION ON VARIABLES,
# CONSTANTS, OPERATORS, AND EXPRESSIONS USED IN C PROGRAMMING.
```

```
#
# THE LESSON CURRENTLY CONSISTS OF FIVE TOPICS.
```

```
#
# The Lesson Breakdown Is As Follows:
```

```
#
# Topic 1: Variables & Constants I - This topic is the first of two that
# covers the declaration and use of variables and constants in
# C programming. (Approx. time = 10 min.)
```

```
#
# Topic 2: Variables & Constants II - This topic is the second of two that
# covers the declaration and use of variables and constants in C
# programming. (Approx. time = 5 min.)
```

```
#
# Topic 3: Operators & Expressions I - This topic is the first of two that
# covers the use of the different operators and expressions in
# C programming. (Approx. time = 15 min.)
```

```
#
# Lesson Breakdown Continued:
```

```
#
# Topic 4: Operators & Expressions II - This topic is the second of two that
# covers the use of the different operators and expressions in C
# programming. (Approx. time = 10 min.)
```

```

#
#
# Topic 5: Lesson 2 Test - This is the lesson test over items that have
# been presented in the previous four lesson topics.
# (Approx. time = 5 min.)
#
#

```

```

# TOTAL LESSON TIME IS APPROXIMATELY 45 MINUTES.
#
#

```

```

# I hope that you enjoy it!
#
#

```

```

*****
#
#

```

```

# SELECT THE TOPIC YOU WISH TO TAKE FROM THE FOLLOWING:
#
#

```

```

*****
#
#

```

STATUS	TOPIC #	TOPIC TITLE
@	1	Variables & Constants I
@	2	Variables & Constants II
@	3	Operators & Expressions I
@	4	Operators & Expressions II
@	5	Test Over Lesson 2

```

*****
# NOTE: A "STATUS" OF "+" INDICATES TOPIC SUCCESSFULLY COMPLETED.
#
*****

```

```

11Frame 100 T VARIABLES & CONSTANTS I

```

```

12 *** Data Types ***

```

```

12

```

```

12 In C there are four sets of basic data types that can be used.
12 These four are: Character, Integer, Floating point, and Double-
12 precision floating point.
12

```

```

12 We will cover the character and integer data types in this topic area,
12 and leave floating point and double-precision floating point for the
12 next topic area.
12

```

```

12 I will be discussing the declaration and use of both variables and
12 constants within the context of data type usage.
12

```

```

12 The flow of this topic area will follow the following outline:
12

```

- | | |
|---------------------------|----------------------|
| 12 1. Character Constants | 3. Integer Constants |
| 12 2. Character Variables | 4. Integer Variables |

13B:105

11Frame 105 T

12 *** Variable Names ***

12

12 Before we get too far into this area, we need to set up some rules
12 for naming any variables that we use in our programming.

12

- 12 1. Variable names must begin with a letter.
- 12 2. Variable names are composed of letters and digits.
- 12 3. Variable names must not be C keywords.

12

12 In C, a "letter" is any character in the set {a..z,A..Z,_}, that's
12 all lower and upper case letters as well as the "underline" character.
12 A "digit" is any character in the set {0..9}. A "keyword" is any word
12 in the set:

12

12 {auto, break, case, char, continue, default, do, double, else, entry,
12 extern, float, for, goto, if, int, long, register, return, short,
12 sizeof, static, struct, switch, typedef, union, unsigned, while}

13B:110

11Frame 110 T

12 * Variable Names Continued *

12

12 A few additional facts need to be mentioned about variable names.

12

- 12 1. Upper and lower case names are different. This means that the
12 variable names: answer, Answer, and ANSWER are all different
12 variable names.
- 12 2. Only the first eight characters of a variable name are significant.
12 This means that insert_A1 and insert_A2 are the same variable name.
- 12 3. The number of significant characters may be less than eight for
12 external variables and function names (system dependent).

13B:115

11Frame 115 QM

12Which of the following is "not" a valid variable name?

13A X123

13

13B first_num

13

13C+ 2nd_in_line

13

13D _OUT_

14 Very good!

14 B:120

15ABD No. The correct answer is "C". Variable names must start with a letter.

15 B:120

15E I'm sorry, "E" was not one of your choices.

15 B:115

11Frame 120 T

12 *** Character Constants ***

12
12 A character constant is symbolized as a single character enclosed
12 within single quotation marks.
12
12 For example: 'a'
12
12 The value of a character constant is actually the numeric equivalent
12 of the character as defined by the computer system's character set.
12 Thus, arithmetic operations using characters is possible but the most
12 common use for character constants is for comparative purposes.
12
12 All this may seem confusing, but it really isn't. We will look at an
12 example of the usage of character constants after we take a look at
12 character variables.

13B:125

11Frame 125 T

12 *** Character Variables ***

12 A character variable is declared by the use of the keyword: char

12 For example: char in_char;

12 The character variable "in_char" will now be assigned a one byte
12 storage location in the computer's memory. The value that will
12 be stored in this location depends on the usage of the variable
12 in the program. Let's look at a couple examples that should help
12 you understand both character variables as well as character con-
12 stants.

12 The statement: in_char = 'a'; assigns the ASCII value 97 (decimal)
12 to the character variable location identified by "in_char" in memory.

12 Note: ASCII values range from 0 thru 127 (decimal) and can be found
12 in most good programming books.

13B:130

11Frame 130 T

12 * Character Variables Continued *

12 Every character on the keyboard has an ASCII numeric equivalent.
12 (By the way, ASCII stands for American Standard Code for Information
12 Interchange.) There are, however, several characters that are hidden.
12 These characters can be represented by character constants by using
12 character escape sequences that start with a backslash (\).

12 Some of the more common character escape sequences follow:

12 \b (backspace)
12 \n (new line)
12 \f (form feed)
12 \r (carriage return)
12 \\ (backslash)
12 \' (single quotation)
12 \### (### = an octal value)

13B:135

11Frame 135 T

12 * Character Variables Continued *

12

12 An example of how you would declare a character variable using one of
12 the special character escape sequences as a character constant is as
12 follows:

12

12 char back_space = '\b';

12

12 This statement assigns the ASCII value 8 (decimal) to the character
12 variable location identified by "back_space" in memory.

12

12 An equivalent way to declare the variable "back_space" is as follows:

12

12 char back_space = '\010'; OR char back_space = '\10';

12

12 In both statements, the character variable location "back_space" is
12 assigned the value 10 (octal) which is equivalent to 8 (decimal).

13B:140

11Frame 140 QP

12In the statement: char input_char = 't';

12

12input_char is called a character variable and 't' is called a
12character constant. (True or False)

13Y

14 That's right.

14 B:145

15 Wrong. That is a true statement.

15 B:145

11Frame 145 QM

12Which one of the following characters is used to identify a special character
12escape sequence?

13A \$

13

13B @

13

13C &

13

13D+ \

13

13E #

14 Very good, you're so right.

14 B:150

15ABCE No. Answer "D" is the correct answer.

15 B:150

11Frame 150 T

12 *** Character Constants and Variables Summary ***

12

12 So far in this topic area you have seen rules over selecting variable
12 names, a description and examples of character constants, a description
12 an examples of character variables, and a description and examples of

12 how to declare special characters.

12

12 In the remainder of this topic area we will look at integer constants

12 and integer variables.

12

12 So let's get to it ...

13B:155

11Frame 155 T

12 *** Integer Size ***

12

12 The first thing we need to cover when talking about integers is the

12 size of a number that can be used. In C we can normally use integers

12 in the range: -32,768 thru +32,767

12

12 If it is necessary to use a number outside this range, C provides a

12 way to accomplish this.

12

12 The use of an "unsigned" integer provides for use of numbers in the

12 range: 0 thru 65,535

12

12 The use of a "long" integer will provide for use of numbers in the

12 range: -2,000,000,000 thru +2,000,000,000

12

12 The way to identify which size you are using will be explained.

13B:160

11Frame 160 T

12 *** Integer Constants ***

12

12 An integer constant can be expressed in one of three ways. It can

12 be decimal, octal, or hexadecimal. Also, each of these can be either

12 a "short" or "long" integer.

12

12 Decimal integer constants are represented by such numbers as: 238,

12 45920, and -72. Note that embedded commas are not used. 45,920

12 would be wrong.

12

12 Octal integer constants are represented by such numbers as: 089, 0150,

12 and 014. Note that "octal" numbers all have a leading "zero".

12

12 Hexadecimal integer constants are represented by such numbers as: 0x8F

12 0X9f, 0X2A, and 0x7b. Note that lower case and upper case can be used

12 and "hexadecimal" numbers all have a leading "zero x".

13B:165

11Frame 165 T

12 * Integer Constants Continued *

12

12 As I mentioned, integer constants can also be either a "short" or

12 "long" integer. An integer will be stored as a "short" integer unless

12 you indicate otherwise. There is, of course, exceptions to the rule.

12 For example, if you specify an integer that is larger than 32767, then

12 it will be stored as a long integer.

12

12 The way to indicate that an integer is to be stored as a "long" integer
12 is to follow the number with the letter "L". Here are a few examples.

12
12 Decimal: 5987L, and 367L
12 Octal: 04689L, and 0824L
12 Hexadecimal: 0X2A5F4L, and 0x6FDAL

12 Note: A lowercase letter "l" may be used, but may be very confusing.

13B:170

11Frame 170 QM

12Which of the following best describes the integer constant 073564L ?

13A Decimal

13

13B Long Decimal

13

13C Octal

13

13D+ Long Octal

13

13E Hexadecimal

14 Right.

14 B:175

15AE No. The "0" (zero) in front makes it an "octal" and the "L" makes it a
15 "long" integer. I think you need to review this material.

15 B:160

15B No. The "0" (zero) in front makes it an "octal" number.

15 B:175

15C No. The "L" after the number makes it a "long" octal number.

15 B:175

11Frame 175 T

12 *** Integer Variables ***

12

12 An integer variable is declared by the use of the keyword: int

12

12 For example: int index;

12

12 The integer variable index will now be assigned a 16 bit storage
12 location in the computer's memory.

12

12 Let's look at a couple examples that should help you understand
12 both integer variables as well as integer constants.

12

12 The statement: number_in = 212; assigns the integer constant value
12 212 (decimal) to the integer variable location identified by "number_in"
12 in memory.

13B:180

11Frame 180 T

12 * Integer Variables Continued *

12

12 When declaring an integer variable, you have the option of specifying
12 whether the variable is to be a "short", "long", or "unsigned" variable.

12

```

12 The way to indicate which of these an integer variable will be is by
12 using the keywords short, long, or unsigned. Here are some examples:
12
12 short int index_1; -----> Or Just -----> short index_1;
12
12 long int index_2; -----> Or Just -----> long index_2;
12
12 unsigned int index_3; -----> Or Just -----> unsigned index_3;
12
12 Note: When using these keywords, use of "int" is optional.
13B:185
11Frame 185 QF
12 When declaring an integer variable, you only have the option of specifying
12 the variable as being either "short" or "long". (True or False)
13N
14 That's right. You can also specify it as being "unsigned".
14 B:190
15 Wrong. You can also specify it as being "unsigned".
15 B:190
11Frame 190 T
12 * Integer Variables Continued *
12
12 One last word on integers.
12
12 Although C has the capability of specifying different size storage
12 locations, this capability is limited by the specific compiler and
12 system you are using. Please check to see if your compiler and
12 system treat integers as described here.
13B:195
11Frame 195 T
12 *** Topic Summary ***
12
12 In this topic area we have looked at a description and examples of
12 character constants, character variables, integer constants, and
12 integer variables. Also we covered variable names and special
12 characters.
12
12 In the next topic area we will continue to discuss constants and
12 variables by looking at floating point and double-precision float-
12 ing point data types.
12
12
12 *** This concludes this topic area. ***
13END
21Frame 300 T VARIABLES & CONSTANTS II
22 *** Data Types ***
22
22 We learned in the last topic area that there are four basic data
22 types used in C. These four are: Character, Integer, Floating
22 point, and Double-precision floating point.
22
22 We covered the character and integer data types in the last topic area,

```

22 so we will cover floating point and double-precision floating point in
22 this topic area.

22 I will be discussing the declaration and use of both variables and
22 constants within the context of data type useage.

22 The flow of this topic area will follow the following outline:

- 22 1. Floating Point and Double-precision Floating Point Constants
- 22 2. Floating Point Double-precision Floating Point Variables

23B:305

21Frame 305 T

22 *** Floating Point and Double-precision Floating Point Constants ***

22 Floating point numbers are just numbers that have two parts instead of
22 one, as in the case of an integer. You can think of a floating point
22 number as having an integer, or whole part, and a fractional part.

22 These two parts are seperated by a decimal point.

22 Examples of floating point numbers: 67.32, 2583.1, and 2.4592

22 How "precise" a number is has an effect on calculations preformed using
22 a stored number. Thus, the precision of a number may be very important
22 within your program. C stores all floating point constants as double
22 precision. This means that a large number of significant digits are
22 stored to represent the number and hence, gives better precision in any
22 calculations preformed involving the number.

23B:310

21Frame 310 T

22 * FP & DFPF Constants Continued *

22 Another way of representing floating point numbers is through the use
22 of "scientific notation". The following are examples of the use of
22 scientific notation for floating point constants:

22 4.67E3 <or> 4.67e3 = 4670.0

22 .9834E2 = 98.34

22 345.0e6 = 345000000.0

22 -2.8473E5 = -284730.0

22 4.67E-3 <or> 4.67e-3 = .00467

22 .9834E-2 = .009834

22 345.0e-6 = .000345

22 -2.8473E-5 = -.000028473

22 Note: The "E" can be upper or lower case ("e").

23B:315

21Frame 315 QM

22Which of the following is "not" an example of a floating point constant?

23A 4670.0

23

```

23B .9834e-2
23
23C+ 345
23
23D -2583.1
23
23E -67.9E3
24 Correct.
24 B:320
25ABDE Wrong. Answer "C" is an "integer".
25 B:320
21Frame 320 T
22 *** Floating Point and Double-precision Floating Point Variables ***
22
22 In C, floating point variables are declared using the keyword "float",
22 and double precision floating point numbers are declared using the
22 the keyword "double".
22
22 Here are some examples:
22
22 float var_1;           | double var_1;
22 float var_2, var_3;    | double var_2, var_3;
22
22 The following illustrates the use of floating point constants and
22 variables.
22
22 float var_1 = 451.29    <or> float var_1 = 4.5129E2
22 double var_2 = 23975.5619 <or> double var_2 = 2.397535619e4
23B:325
21Frame 325 T
22 * FP & DPFP Variables Continued *
22
22 To reiterate, the use of "double" allows for the storing of a greater
22 number of significant digits to represent a given number. Thus, more
22 precision is gained in calculations involving the number.
22
22 Another way of achieving the precision of a double precision variable
22 is with the keyword "long". The following two statements have the
22 same effect:
22
22 double var_1; >>>>> OR >>>>> long float var_1;
23B:330
21Frame 330 QP
22In the statement double var_one = 419.9253; the keyword "double" is used
22to indicate that variable "var_one" is to be stored as a "double precision
22floating point" number. (True or False)
23Y
24 Very good.
24 B:335
25 No. That statement is true.
25 B:335
21Frame 335 T

```

22 *** Topic Summary ***

22

22 In this topic area we have looked at a description and examples of
22 floating point and double precision floating point constants, and
22 floating point and double precision floating point variables.

22

22 In the next topic area we will begin a discussion of operators and
22 expressions and their use in C.

22

22

22 *** This concludes this topic area. ***

23END

31Frame 500 T OPERATORS & EXPRESSIONS I

32 *** Introduction ***

32

32 In this and the next topic area we will be discussing operators and
32 their use in expressions.

32

32 This first topic area will cover the following:

32

- 32 1. Arithmetic Operators
- 32 2. Increment & Decrement Operators
- 32 3. Assignment Operators

32

32

32 Let's get started ...

33B:505

31Frame 505 T

32 *** Arithmetic Operators ***

32

32 The arithmetic operators are represented by the following:

32

32 Addition (+), Subtraction (-), Multiplication (*), Division (/),
32 Modulus (%), and the Unary minus (~).

32

32 The first four in this list are probably the most familiar to you so
32 I will only give one example of their use in an expression.

32

32 Addition: $a + b$ (adds b to a)

32

32 Subtraction: $a - b$ (subtracts b from a)

32

32 Multiplication: $a * b$ (multiplies a by b)

32

32 Division: a / b (divides a by b)

33B:510

31Frame 510 T

32 * Arithmetic Operators Continued *

32

32 The modulus operator can be used only with integer (int) data types.

32

32 The action performed by this operator is one of returning the remainder

32 after a division operation. For example, in the statement:
32
32 Answer = 15 % 2; the value stored in "Answer" would be 1, since 15
32 divided by 2 is 7 with a remainder of 1. Likewise:
32
32 Result = 150 % 15; produces a value of 0 in "Result", since 15 divides
32 150 evenly.

33B:515

31Frame 515 T

32 * Arithmetic Operators Continued *

32
32 The unary minus operator is used to change the sign of the operand it
32 operates on.

32
32 The action performed by this operator is one of returning the negative
32 of the value of the operand. For example, in the statement:

32
32 Answer = -x_value; the value stored in "Answer" would be the negative
32 of the value stored in "x_value". For instance:

32
32 If the value stored in "x_value" is 385, then the value stored in the
32 variable "Answer" would be -385. Likewise, if the value in "x_value"
32 were -952, then "Answer" would contain the value 952.

32
32 Note: C does not have a unary plus operator.

33B:520

31Frame 520 QM

32Which of the following is the value that will be assigned to the variable
32"Answer" after execution of the statement: Answer = 27 % 12; ?

33A 2.25

33

33B+ 3

33

33C .25

33

33D 2

34 Very good.

34 B:525

35ACD No. The modulus operator returns the "remainder" of "integer" division,
35 therefore answer "B" is correct.

35 B:525

35E I'm sorry, "E" was not one of your choices.

35 B:520

31Frame 525 T

32 *** Increment & Decrement Operators ***

32

32 The increment and decrement operators are represented by the following:

32

32 Increment (++) and Decrement (--)

32

32 These two operators can be used in either "prefix" or "postfix"
32 notation.

32
 32 "Prefix" notation results in the variable being incremented or decre-
 32 mented before its value is taken. Whereas, "postfix" notation results
 32 in taking the variable value before it is incremented or decremented.
 32
 32 Let's take a look at each of these operators and see how "prefix" and
 32 "postfix" affects them.
 33B:530
 31Frame 530 T
 32 * Increment Operator *
 32
 32 In the statement: `x_value = x_value + 1;` the value of `x_value` is
 32 incremented by 1 and restored in the memory location identified by
 32 the variable "`x_value`". This is a valid statement in C, but C also
 32 allows a shorthand way of doing the same thing. In this shorthand
 32 notation, the statement would be written as `x_value++;` Thus:
 32
 32 `x_value = x_value + 1;` and `x_value++;` are equivalent statements.
 32
 32 The above example also demonstrates the use of "postfix" notation.
 32 The same result could have been obtained by using "prefix" notation.
 32 If you were to use the statement: `++x_value;` the stored value of
 32 "`x_value`" would have been incremented by 1 as it was using the other
 32 two statements. Where's the difference then? Well, let's look at
 32 another example and see if it becomes clearer.
 33B:535
 31Frame 535 T
 32 * Increment Operator Continued *
 32
 32 If we assign the value of 10 to the variable "`x_value`" using the
 32 statement: `x_value = 10;` and then perform some arithmetic operation
 32 using the variable "`x_value`" and the increment operator, what would be
 32 the result?
 32
 32 Well, it all depends on whether you use "prefix" or "postfix" notation.
 32
 32 If we perform the statement: `Result = ++x_value;` then the value
 32 stored in "`Result`" is 11, and "`x_value`" is incremented to 11, but
 32 if we perform the statement: `Result = x_value++;` then the value
 32 stored in "`Result`" is 10, and "`x_value`" is incremented to 11.
 32
 32 As you can see, this can be confusing until you get used to the idea.
 33B:540
 31Frame 540 QP
 32The placement of the "increment" operator (either before or after the
 32variable) has no effect on the outcome of statement execution.
 32(True or false)
 33N
 34 Right.
 34 B:545
 35 Wrong. `variable++` and `++variable` will produce different results depending
 35 on how and when they are used.

35 B:545

31Frame 545 T

32 * Decrement Operator *

32

32 In the statement: `y_value = y_value - 1;` the value of `y_value` is
32 decremented by 1 and restored in the memory location identified by
32 the variable "`y_value`". This is a valid statement in C, but again
32 C has a shorthand way of doing the same thing. In this shorthand
32 notation, the statement would be written as `y_value--;` Thus:

32

32 `y_value = y_value - 1;` and `y_value--;` are equivalent statements.

32

32 The above example again demonstrates the use of "postfix" notation.

32 The same result could have been obtained by using "prefix" notation.

32 If you were to use the statement: `--y_value;` the stored value of

32 "`y_value`" would have been decremented by 1 as it was using the other

32 two statements. Let's again look at an example showing the difference

32 between using "prefix" and "postfix" notation.

33B:550

31Frame 550 T

32 * Decrement Operator Continued *

32

32 If we assign the value of 15 to the variable "`y_value`" using the
32 statement: `y_value = 15;` and then perform some arithmetic operation
32 using the variable "`y_value`" and the decrement operator, what would be
32 the result?

32

32 Well, again it all depends on whether you use "prefix" or "postfix"
32 notation.

32

32 If we perform the statement: `Answer = --y_value;` then the value
32 stored in "`Answer`" is 14, and "`y_value`" is decremented to 14, but
32 if we perform the statement: `Answer = y_value--;` then the value
32 stored in "`Answer`" is 15, and "`y_value`" is decremented to 14.

32

32 Remember: prefix - value taken second, postfix - value taken first.

33B:555

31Frame 555 QM

32Which of the following represents the contents of variables "`Answer`" and

32"`y_value`" after execution of the statement: `Answer = 25 + (--y_value);`

32given the initial value of "`y_value`" is 10?

33A+ `Answer = 16` and `y_value = 9`

33

33B `Answer = 15` and `y_value = 9`

33

33C `Answer = 16` and `y_value = 10`

33

33D `Answer = 15` and `y_value = 10`

34 You are correct.

34 B:560

35BCD Wrong. Choice "A" is correct.

35 B:560

35E I'm sorry, "E" was not one of your choices.

35 B:555

31Frame 560 T

32 *** Assignment Operators ***

32

32 The assignment operators are represented by the following:

32

32 Equal (=), and Operation equal (op=), where the "operation" is one of the binary operators.

32

32 We have already seen how the first assignment operator is used.

32 As an example we have a statement such as: Answer = 25;

32

32 In this example, the equal assignment operator is used to place the value of 25 in the memory location represented by the variable "Answer".

32

32 This assignment is done "right to left", so it is possible to make

32

32 several assignments using one statement. For example:

32

32 a_val = b_val = c_val = 0; will set all the named variables to zero.

33B:565

31Frame 565 T

32 * Assignment Operators Continued *

32

32 The "operation equal" operators are really nothing more than a short-hand method of writing a statement that involves doing some operation on a variable and storing the result back into that variable's memory location. For example, in the statement:

32

32 x_value = x_value + 25; 25 is added to the value of x_value and the result is stored in the memory location represented by x_value.

32

32 C provides a way to accomplish this in a shorter statement (although the one above is also valid). An equivalent C statement would be:

32

32 x_value += 25;

32

32 All operations on the right will be done before the operation identified in front of the "=" sign.

33B:570

31Frame 570 T

32 * Assignment Operators Continued *

32

32 That last statement is an important one. For example, in the statement

32

32 a_val *= b_val + c_val; you will get a different result if the statement were evaluated as: a_val = (a_val * b_val) + c_val;

32

32 To eliminate this problem, C will evaluate the statement according

32

32 to the following rule:

32

32 left_variable = (left_variable) "op" (right_expression);

32

32 Our example will be evaluated as: a_val = (a_val) * (b_val + c_val);

33B:575

31Frame 575 QP

32The result stored in "x_value" after execution of the statement:

32x_value -= 35 + 20; given an initial value for "x_value" of 100,

32would be 45. (True or False)

33Y

34 Right.

34 B:580

35 Wrong.

35 B:580

31Frame 580 T

32 *** Topic Summary ***

32

32 In this topic area we have looked at a description and examples of
32 arithmetic, increment & decrement, and assignment operators.

32

32 In the next topic area we will continue our discussion of operators
32 and expressions and their use in C.

32

32

32 *** This concludes this topic area. ***

33END

41Frame 700 T OPERATORS & EXPRESSIONS II

42 *** Introduction ***

42

42 In this topic area we will be continue discussing the use of operators
42 in expressions that we started in the last topic area.

42

42 This second topic area on this subject will cover the following:

42

42 1. Relational Operators

42 2. Logical Operators

42 3. Bitwise Logical Operators

42 4. Negation Operator

42 5. Conditional Operator

42

42

42 Let's get started ...

43B:705

41Frame 705 T

42 *** Relational Operators ***

42

42 Relational operators are used within a program in order to compare
42 one or more data values. The relational operators are represented
42 by the following:

42

42 Equality (==), Inequality (!=), Greater than (>), Greater than or
42 equal to (>=), Less than (<), and Less than or equal to (<=).

42

42 Expressions involving these operators are evaluated as being either
42 "true" or "false". If an expression is "true" then the expression
42 has a value of 1 (one), if it is "false" then the value is 0 (zero).

```

42
42   Let's take a look at an example to help make this clear.
43B:710
41Frame 710 T
42   * Relational Operators Continued *
42
42   For our example let's compare two variables:
42
42   var_1 >= var_2 is an expression that has a value of either "true"
42   or "false". If var_1 is indeed greater than or equal to var_2, then
42   the expression is "true" and has a value of 1 (one). Likewise, if
42   var_1 is not greater than or equal to var_2, then the expression is
42   "false" and has a value of 0 (zero).
42
42   In order to give this evaluation meaning, it must be somehow used in
42   a valid C statement. An easy to understand example is:
42
42   var_flag = (var_1 >= var_2);
42
42   "var_flag" will be assigned either 1 or 0 depending on the evaluation
42   of the expression: var_1 >= var_2
43B:715
41Frame 715 T
42   * Relational Operators Continued *
42
42   The example we looked at was one in which a comparison was made between
42   two variables. This is not the only way to use the relational operators
42   as you can well imagine.
42
42   Some of the more common situations that relational operators are used
42   in include: comparing array values, checking for "end of file",
42   controlling function calls, and controlling statement execution.
42
42   Relational operators have a lower precedence than arithmetic operators,
42   and assignment operators have lower precedence than relational operators
42   thus, the statement: val_one = val_2 != val_3 - 5; will be evaluated
42   as: val_one = (val_2 != (val_3 - 5)); The final value of either 1 or
42   0 will eventually be stored in the memory location represented by the
42   variable "val_one".
43B:720
41Frame 720 QM
42Which of the following is "not" a relational operator?
43A ==
43
43B !=
43
43C >=
43
43D <
43
43E+ ++
44 Right.

```

```

44 B:725
45ABCD Wrong. That is one of the relational operators, choice "E" is not.
45 B:725
41Frame 725 T
42   *** Logical Operators ***
42
42   Logical operators are also called logical connectives and they are
42   used to combine expressions being used for comparison. The logical
42   operators are represented by the following:
42
42   Logical AND (&&)      and      Logical OR (||)
42
42   Expressions involving these operators are evaluated as being either
42   "true" or "false". If an expression is "true" then the expression
42   has a value of 1 (one), if it is "false" then the value is 0 (zero).
42
42   Let's take a look at an example.
43B:730
41Frame 730 T
42   * Logical Operators Continued *
42
42   In the expression: in_char == 'y' || in_char == 'n', the value of the
42   expression can once again have a value of either 1 or 0 depending on
42   whether the expression is "true" or "false".
42
42   In order to give this evaluation meaning, it must be somehow used in
42   a valid C statement. An example is:
42
42   valid_resp = (in_char == 'y' || in_char == 'n');
42
42   "valid_resp" will be assigned either 1 or 0 depending on the evaluation
42   of the expression: in_char == 'y' || in_char == 'n'
42
42   In other words, valid_resp will be 1 if in_char equals either y OR n,
42   or 0 if in_char equals anything else.
43B:735
41Frame 735 T
42   * Logical Operators Continued *
42
42   Logical operators have a lower precedence than relational operators,
42   and assignment operators have lower precedence than logical operators
42   thus, the statement:
42
42   result_val = val_1 < val_2 && val_3 != val_4;
42
42   is evaluated as: result_val = ((val_1 < val_2) && (val_3 != val_4));
42
42   The final value of either 1 or 0 will eventually be stored in the
42   memory location represented by the variable "result_val".
43B:740
41Frame 740 QP
42In the statement: True_Response = choice == 't' || choice == 'T';

```

42 the value of "True_Response" will be set to 1, only if both "choice == 't'"
42 and "choice == 'T'" are true. (True or False)

43N

44 That's right.

44 B:745

45 Wrong. Only one of the expressions has to be true when using the "OR"
45 logical operator.

45 B:745

41Frame 745 T

42 *** Bitwise Logical Operators ***

42

42 The use of bitwise logical operators is beyond the scope of this
42 course. However, I feel that their existence should be mentioned.

42

42 Bitwise logical operators are represented by the following:

42

42 Bitwise AND: &

42 Bitwise inclusive OR: |

42 Bitwise exclusive OR: ^

42 Left shift: <<

42 Right shift: >>

42 Unary one's complement: ~

43B:750

41Frame 750 T

42 *** Negation Operator ***

42

42 The negation operator is a unary NOT operator. It is used to convert
42 or reverse the value of the operand it appears in front of.

42

42 The exclamation point (!) is used for this operator.

42

42 For example, in the expression: !(val_one < 30) if the value of the
42 inner expression is "true" then the value of the entire expression is
42 "false", and vice versa.

42

42 The parentheses, in this case, are necessary since the negation oper-
42 ator has a higher precedence than relational operators.

42

42 A statement using the negation operator would look something like this:

42

42 control_flag = ! found_flag;

43B:755

41Frame 755 T

42 *** Conditional Operator ***

42

42 The conditional operator is what is called a "ternary" operator.

42

42 What this means is that the operator acts upon three operands. The
42 effect it has is very similar to an "if-else" control statement.

42

42 The conditional operator is represented by a question mark and colon.

43B:760

41Frame 760 T

42 *** Conditional Operator Continued ***

42

42 For example, in the statement:

42

42 new_val = val_1 == 1 ? new_val = 25 : new_val = 30;

42

42 The final value of "new_val" depends on the value of "val_1". If
42 val_1 is equal to 1, then the expression "val_1 == 1" is true and
42 the expression "new_val = 25" will be executed, else if "val_1 == 1"
42 is false, then the expression "new_val = 30" will be executed. The
42 value of new_val will be stored in the memory location represented
42 by the variable "new_val" because of the assignment operator "="
42 after the variable "new_val" on the left side of the statement.

43B:765

41Frame 765 QP

42In the statement: val_1 = test_val ? val_1 = 1 : val_1 = 0;

42if test_val equals 1, then the value of val_1 will be 1. (True or False)

43Y

44 Right.

44 B:770

45 Wrong. "val_1 = 1" would be executed, thus setting "val_1" equal to 1.

45 B:770

41Frame 770 T

42 *** Lesson Two Summary ***

42

42 Well, that about does it for lesson number two. If you have seen the
42 four subject topics in this lesson, you should now be ready to take
42 the final test. If you feel that you don't understand something well
42 enough to pass the test, please retake the topic that is giving you
42 problems.

42

42 Topic # |----- Subject covered -----|

42

42 1 Character & interger constants and variables.

42

42 2 Real & double precision real constants and variables.

42

42 3 Arithmetic, increment & decrement, and assignment operators.

42

42 4 Relational, logical, negation, and conditional operators.

43END

51Frame 900 TT TEST OVER LESSON 2

52 Welcome to the final test of lesson two. This test consists of ten
52 questions over material presented in the previous four topic areas.

52

52 In order to successfully complete this lesson, you must achieve a
52 minimum score of 70% (seven out of ten questions correct).

52

52 If you miss a question, the correct answer will not be shown. It is
52 up to you to research the correct answer.

52

52 Well, enough said. Let's get on with it. Good luck!

53B:905

51Frame 905 QM

521. Which of the following is not a valid variable name?

53A x1_y2

53

53B+ int

53

53C _IN_

53

53D var

54 Right. (1,105)

54 B:910

55ACD Wrong. (1,105)

55 B:910

55E "E" was not one of your choices.

55 B:905

51Frame 910 QP

522. A character constant is symbolized as a single character enclosed within single quotation marks. (True or False)

53Y

54 Right. (1,120)

54 B:915

55 Wrong. (1,120)

55 B:915

51Frame 915 QM

523. Which one of the following characters is used to identify a special character escape sequence?

53A #

53

53B +

53

53C+ \

53

53D %

54 Right. (1,130)

54 B:920

55ABD Wrong. (1,130)

55 B:920

55E "E" was not one of your choices.

55 B:915

51Frame 920 QP

524. In the statement: double var_one = 358.8204; the keyword "double" is used to indicate that variable "var_one" is to be doubled in value before being stored in the memory location represented by "var_one". (True or False)

53N

54 Right. (2,315)

54 B:925

55 Wrong. (2,315)

55 B:925

51Frame 925 QM

525. Which of the following is "not" an arithmetic operator?

53A +

53

53B *

53

53C /

53

53D+ =

54 Right. (3,505)

54 B:930

55ABC Wrong. (3,505)

55 B:930

55E "E" was not one of your choices.

55 B:925

51Frame 930 QM

526. Which of the following is the value that will be assigned to the variable

52"Answer" after execution of the statement: Answer = 22 % 5;

53A+ 2

53

53B 4.4

53

53C 4

53

53D .4

54 Right. (3,510)

54 B:935

55BCD Wrong. (3,510)

55 B:935

55E "E" was not one of your choices.

55 B:930

51Frame 935 QP

527. The placement of the "increment" (++) or decrement (--) operators, with

52respect to the variable they operate on, never has an effect on the outcome

52of statement execution.

53N

54 Right. (3,530 & 545)

54 B:940

55 Wrong. (3,530 & 545)

55 B:940

51Frame 940 QP

528. The result stored in "Answer" after execution of the statement:

52Answer *= 10 + 10; given an initial value for "Answer" of 10,

52would be 220. (True or False)

53N

54 Right. (3,555)

54 B:945

55 Wrong. (3,555)

55 B:945

51Frame 945 QM

529. Which of the following is "not" a relational operator?

53A >=

53

53B <=
 53
 53C ==
 53
 53D+ +=
 54 Right. (4,705)
 54 B:950
 55ABC Wrong. (4,705)
 55 B:950
 55E "E" was not one of your choices.
 55 B:945
 51Frame 950 QM
 5210. Which of the following represent the logical operators "OR" and "AND"?
 53A !! and ~
 53
 53B ~ and &&
 53
 53C ## and !!
 53
 53D+ !! and &&
 53
 53E @@ and ++
 54 Right. (4,720)
 54 B:955
 55ABCE Wrong. (4,720)
 55 B:955
 51Frame 955 T
 52 *** End of Lesson Material ***
 52
 52 This marks the end of lesson number two. I hope that it was of some
 52 benefit to you. I am looking forward to seeing you in lesson number
 52 three. I hope that you didn't have too much trouble with the material
 52 presented in this lesson. If you did, please voice your comments to
 52 your training monitor who will in turn contact the CAI Plans Branch
 52 at Keesler AFB, MS.
 52
 52 Well, let's take a look at how you did with the test ...
 53END

File "LESSON3"

```

#      WW      WW  EEEEEEEE  LL          CCCCCC  000000  MMM  MMM  EEEEEEEE
#      WW WW WW  EE          LL          CC          00  00  MMM  MMM  EE
#      WW WW WW  EEEEE  LL          CC          00  00  MM MM MM  EEEEE
#      WWW  WWW  EE          LL          CC          00  00  MM MM MM  EE
#      WWW  WWW  EEEEEEEE  LLLLLLLL  CCCCCC  000000  MM      MM  EEEEEEEE
#
#
#              TTTTTTTTTT  00000000
#              TT          00      00
#              TT          00      00
#              TT          00      00
#              TT          00000000
#
#
#      LL          EEEEEEEE  SSSSSS  SSSSSS  000000  NN      NN  3333333
#      LL          EEEEEEEE  SSS SSS  SSS SSS  00000000  NNN  NN  33333333
#      LL          EE          SSS          SSS          00  00  NNNN  NN  33
#      LL          EEEEE  SSSS          SSSS          00  00  NN NN NN  333
#      LL          EEEEE          SSSS          SSSS          00  00  NN NN NN  33
#      LL          EE          SSS          SSS          00  00  NN  NNNN  3  33
#      LLLLLLLL  EEEEEEEE  SSS SSS  SSS SSS  00000000  NN      NN  33333333
#      LLLLLLLL  EEEEEEEE  SSSSSS  SSSSSS  000000  NN      NN  333333
#
#
# THE LESSON YOU ARE ABOUT TO TAKE CONTAINS INFORMATION ON PROGRAM CONTROL
# STATEMENTS USED IN THE C PROGRAMMING LANGUAGE.
#
# THE LESSON CURRENTLY CONSISTS OF FIVE TOPICS.
#
# The Lesson Breakdown Is As Follows:
#
# Topic 1: If, If-Else, Nesting, and Switch - This topic gives descriptions
# of the structure and use of the If and If-Else control statements
# and how to "nest" these statements. Also covered in this topic
# is the Switch control structure. (Approx. time = 15 min.)
#
# Topic 2: Loops (While, For, and Do-While) - This topic discusses the struc-
# ture and use of loop statements. (Approx. time = 15 min.)
#
# Topic 3: Break and Continue Statements - This topic gives a description of
# the Break and Continue statements and how and when they are used.
# (Approx. time = 10 min.)
#
# Lesson Breakdown Continued:
#
# Topic 4: Goto statement and Labels - This topic gives a description of the
# Goto statement and the use of labels within a C program.
# (Approx. time = 5 min.)

```

```

#
#
# Topic 5: Lesson 3 Test - This is the lesson test over items that have
# been presented in the previous four lesson topics.
# (Approx. time = 5 min.)
#
#
# TOTAL LESSON TIME IS APPROXIMATELY 50 MINUTES.
#
#
# I hope that you enjoy it!
#
!
*****
*
*          SELECT THE TOPIC YOU WISH TO TAKE FROM THE FOLLOWING:
*
*****
*
*          STATUS          TOPIC #          TOPIC TITLE
*          -----          -
*
@          1          If, If-Else, Nesting, and Switch
*
@          2          Loops (While, For, and Do-While)
*
@          3          Break and Continue Statements
*
@          4          Goto statement and Labels
*
@          5          Test Over Lesson 3
*
*****
*
*   NOTE: A "STATUS" OF "+" INDICATES TOPIC SUCCESSFULLY COMPLETED.
*
*****
11Frame 100 T IF, IF-ELSE, NESTING, SWITCH
12   *** Control Statements ***
12
12   Control statements are used in programming languages to provide a
12   means of altering the "normal" flow of the program.
12
12   Without the use of control statements, program execution would proceed
12   in a sequential fashion starting with the first executable statement
12   and ending with the last executable statement. In most cases this is
12   not the desired way in which the programmer wants the program to exe-
12   cute. Thus, the need and capability for imposing control over program
12   execution using program control statements.
12
12   The control statements that we will be looking at in this topic are
12   the "if", "if-else", and "switch". We will also cover "nesting" of
12   "if" statements.
12
12   Let's get started.

```

13B:105

11Frame 105 T

12 *** If Statement ***

12

12 The "if" statement is used to control the execution of a statement or
12 statements by testing an expression. The expression is checked to see
12 if it is "true" (non zero) or "false" (zero). If the statement is in-
12 deed "true", then the statement (or statements) following the "if" is
12 executed. If the expression is "false", then the next sequential
12 statement is executed.

12

12 The structure of the basic "if" statement is as follows:

12

12 if(test_expression)
12 statement_to_be_executed;

12

12 next_sequential_statement;

12

12 Let's take a look at an actual example of the basic "if" statement.

13B:110

11Frame 110 T

12 * If Statement Continued *

12

12 For this example let tax_val, high_tax, and tax_rate be of type "int".

12

12 if(tax_val >= 10)
12 high_tax++;

12

12 tax_rate = tax_val / 100;

12

12 In this example, the expression "tax_val >= 10" is tested. If the
12 value of "tax_val" is greater than or equal to 10, then the statement
12 "high_tax++;" is executed, otherwise program execution continues with
12 the statement "tax_rate = tax_val / 100;"

12

12 By now you should be asking yourself: "How does the compiler know what
12 statement is associated with the 'if' statement?" The answer of course
12 is quite simple. Let's clear up the question and expand the "if".

13B:115

11Frame 115 T

12 * If Statement Continued *

12

12 The example we just looked at could just as easily have been written as:

12

12 if(tax_val >= 10)
12 high_tax++;
12 tax_rate = tax_val / 100;

12

12 This is confusing to the programmer, but not to the compiler. When
12 the "if" statement is encountered, the next sequential statement is
12 the only one that is associated with it. Therefore, only "high_tax++;"
12 is subject to conditional execution. The statement "tax_rate = tax_val

12 / 100;" will be executed no matter what the result of the "if" test is.

12

12 This brings up the question of how do we provide for the execution of
12 several statements after the "if" statement? Let's take a look.

13B:120

11Frame 120 T

12 * If Statement Continued *

12

12 If it is desired to have a group of statement's execution controlled
12 by an "if" statement, then you must use braces "{}" to form a "block"
12 of one or more statements to be conditionally executed. For example:

12

12 if(test_expression)

12 {

12 first_statement;

12 second_statement;

12 .

12 .

12 .

12 last_statement;

12 }

12

12 next_sequential_statement;

13B:125

11Frame 125 QM

12 Which of the following is used to "block" statements into a group to be
12 conditionally executed?

13A ()

13

13B []

13

13C+ { }

13

13D ; ;

14 Correct.

14 B:130

15ABD Wrong. Choice "C" is correct.

15 B:130

15E I'm sorry, "E" was not one of your choices.

15 B:125

11Frame 130 T

12 *** If-Else Control Structure ***

12

12 An option to the "if" statement is the use of an "else".

12

12 I will use the same example as before to illustrate the structure and
12 use of the "if-else" control structure.

12

12 if(tax_val >= 10)

12 high_tax++;

12 else

12 low_tax++;

```

12  tax_rate = tax_val / 100;
12
12  Here, the expression "tax_val >= 10" is tested.  If the value of the
12  "tax_val" is greater than or equal to 10, then "high_tax++;" is execu-
12  ted, otherwise "low_tax++;" is executed before execution continues with
12  the statement "tax_rate = tax_val / 100;"

```

13B:135

11Frame 135 T

```

12  * If-Else Control Structure Continued *

```

```

12
12  Of course you may use a "block" of statements after either or both the
12  "if" or "else" parts of the control structure.  For example:

```

```

12
12  if(test_expression)
12  {
12      statement_1;
12      statement_2;
12  }
12  else
12  {
12      alt_statement_1;
12      alt_statement_2;
12      statement_3;
12  }
12  next_sequential_statement;

```

13B:140

11Frame 140 T

```

12  * If-Else Control Structure Continued *

```

```

12
12  It is often the case that you may need to test more than one expres-
12  sion within an "if-else" structure.  This may be done by using what
12  is called a multi-way decision structure.  I will show you one way
12  to do this using the "if-else" structure now, and later we will see
12  another way using the "switch" structure.  Using "if-else" structure:

```

```

12  if(tax_val >= 10)           ; Using this structure, one of the
12      high_tax++;             ; variables: "high_tax", "low_tax",
12  else if(tax_val <=5)        ; or "medium_tax" will get incremented
12      low_tax++;              ; depending on the value of "tax_val".
12  else                        ; If you didn't want to keep track of
12      medium_tax++;           ; "medium_tax", you could leave off
12  tax_rate = tax_val / 100;    ; the last else and its statement.

```

13B:145

11Frame 145 QP

```

12When using the "if-else" control structure, you are limited to only one
12executable statement for each part of the structure.  (True or False)

```

13N

```

14 Right.  "Blocks" of statements can be defined by the use of braces "{}".

```

14 B:150

```

15 Wrong.  "Blocks" of statements can be defined by the use of braces "{}".

```

15 B:150

11Frame 150 T

12 *** Nesting ***

12
12 Another capability of the "if-else" structure is being able to "nest"
12 other "if" or "if-else" structures within the original "if-else".
12 For example:

```
12 if(test_exp_1)           | In this structure, if "test_exp_1" is true
12     if(test_exp_2)       | then "test_exp_2" is checked and if found
12         statement_one;   | true, then "statement_one" is executed if
12     else                 | however, "test_exp_2" is false, then the
12         alt_statement_one; | "alt_statement_one" is executed. If the
12 else                   | "test_exp_1" was found to be false, then
12     statement_two;      | only "statement_two" would be executed.
12 next_seq_statement;    |
```

12 Note: True is any "non zero" value and false is a "zero" value.
12 Also, "blocks" of statements can be used in the structure.

13B:155

11Frame 155 T

12 * Nesting Continued *

12
12 Caution must be exercised when nesting "if-else" structures. Remember,
12 the "else" part of the "if-else" structure is optional. Thus, it is
12 fairly easy to have an "else" apply to the wrong "if" statement.
12 Let's take a look at an example to show how this can happen.

12
12 For our example, let's say we want to check an expression and if it is
12 true, then we want to check a second expression and if it is true, then
12 we want to execute a statement, but if the second expression is false,
12 we don't want any statement executed. If however, our first expression
12 is false, then we want to execute a different statement. How would we
12 code such a thing? Well, let's give it a try.

13B:160

11Frame 160 T

12 * Nesting Continued *

12
12 At first glance the following seems to do what was described.

```
12 if(test_exp_1)
12     if(test_exp_2)
12         statement_one;
12 else
12     statement_two;
```

12 Even though I indented the code to look like the "else" goes with the
12 first "if", it really goes with the last "if" that doesn't have an
12 "else". Thus, the above code doesn't solve the problem as I stated it.

13B:165

11Frame 165 T

12 * Nesting Continued *

12
12 In order to solve the stated problem, we must use braces to force

12 program execution.

12

12 Compare the code I gave before (left) to the correct code (right).

12

12	if(test_exp_1)	:	if(test_exp_1)
12	if(test_exp_2)	:	{
12	statement_one;	:	if(test_exp_2)
12	else	:	statement_one;
12	statement_two;	:	}
12		:	else
12		:	statement_two;

12

12 As you can see, the execution of the code is greatly affected by the
12 placement of the braces in the "if-else" control structure.

13B:170

11Frame 170 QM

12Given: if(x > 0)

```
12    if(x > 10)
12        x_large = 1;
12    else
12        x_small = 1;
12    else
12        if(x == 0)
12            x_zero = 1;
```

12

12Which of the following would be true if x = -1 ?

13A x_large would be set to 1

13B x_small would be set to 1

13C x_zero would be set to 1

13D+ none of the above

13E A, B, and C would be true

14 Very good.

14 B:175

15ABCE Wrong. All expressions would be false, therefore no statements would
15 be executed.

15 B:175

11Frame 175 T

12 *** Switch Statement ***

12

12 We saw earlier that one way to do multi-way decisions was with the use
12 of several "if-else" statements linked together.

12

12 A common use of such a structure is when you test a variable and depend-
12 ing on its value (as compared to a constant) a statement or group of
12 statements is executed. For example:

12

```
12 if(test_var == 10)
12     statement_to_be_executed;
12 else if (test_var == 15)
12     alt1_statement_to_be_executed;
12 else if (test_var == 20)
12     alt2_statement_to_be_executed;
```

```

12     else
12         default_statement_to_be_executed;
13B:180
11Frame 180 T
12     * Switch Statement Continued *
12
12     In the example, we saw how to use the "if-else" structure to accomplish
12     the testing of one variable and execution of different statements depen-
12     ding on the value of the variable.
12
12     Well, in C we have another way to accomplish the same thing. We can
12     use the "switch" statement. In the "switch" statement each constant
12     value we wish to test the variable against is labeled with the keyword
12     "case". The last statement (following the last "else" in our example)
12     is labeled with the keyword "default".
12
12     Let's take a look at our example again, but this time we will use the
12     "switch" statement structure.
13B:185
11Frame 185 T
12     * Switch Statement Continued *
12
12     switch(test_var) {
12         case 10:
12             statement_to_be_executed;
12             break;
12         case 15:
12             alt1_statement_to_be_executed;
12             break;
12         case 20:
12             alt2_statement_to_be_executed;
12             break;
12         default:
12             default_statement_to_be_executed;
12             break;
12     };
12
12     Note: "test_var" must
12     evaluate to type "int",
12     braces are used, colons
12     are used after each case
12     constant (constant ex-
12     pression), "break" is
12     discussed in topic 3 of
12     this lesson, "break" &
12     "default" are optional,
12     the final semicolon is
12     required since "switch"
12     is really just a "block"
12     type statement, and the
12     order of cases/default
12     is arbitrary.
13B:190
11Frame 190 QP
12Essentially, the "switch" is just a special case of the "if-else" structure,
12and its use is really just "programmer preference". (True or False)
13Y
14 Right. You can do the same thing using the "if-else" structure.
14 B:195
15 Wrong. You can do the same thing using the "if-else" structure.
15 B:195
11Frame 195 T
12     *** Topic Review ***
12
12     In this topic we have looked at the "if" statement, the "if-else" struc-
12     ture, nesting of the "if-else" structure, and the "switch" statement.
12

```

```

12 We have seen many examples of what these statements and structures look
12 like, and how they are used.
12
12 In the next topic area I will describe and show examples of loop state-
12 ments and structures.
12
12 See you there!
12
12 *** This concludes this topic area. ***
13END
21Frame 300 T LOOPS (WHILE, FOR, DO-WHILE)
22 *** Loops ***
22
22 Loops are used in programming languages to provide a way of repeatedly
22 executing a statement or group of statements within the program.
22
22 The way in which a loop is written can vary. The most common reason
22 for this variability is again "programmer preference". Most, if not
22 all, loops can be written using only one of the structures that we
22 will be covering in this topic area.
22
22 The loop control statements and structures that we will be looking at
22 in this topic are the "While", "For", and "Do-While".
22
22 Let's get started.
23B:305
21Frame 305 T
22 *** While Loop ***
22
22 The "while" loop is a two part control structure. The first part is
22 the loop control expression, and the second part is the executable
22 body.
22
22 The loop control expression is a expression that is tested at the be-
22 ginning of the loop and after execution of the body. The loop control
22 expression is "true" whenever it is "non zero" and "false" when it is
22 "zero". Execution of the body will continue until the control expres-
22 sion is "false". If the expression is "false" the first time, then
22 program control will drop to the next sequential program statement.
22
22 The structure of the "while" loop looks like this:
22
22 while (test_expression)      ; Of course, braces can be used to
22 statement_to_be_executed;    ; define a "block" of statements.
23B:310
21Frame 310 T
22 * While Loop Continued *
22
22 Here is an example using the "while" loop control statement:
22
22 sum = 0;

```

```

22  loop_var = 0;
22  while (loop_var == 0) {
22      if(sum < 10)
22          sum += 2;
22      else
22          loop_var++;
22  }
22

```

22 When the "while" is encountered, the test expression is checked and found to be "true", so the loop body is then executed. Execution of the loop body will continue until the loop control expression is no longer true. That will occur, in this example, after 6 iterations.

23B:315

21Frame 315 QP

22 If the loop control expression is "false" the first time it is checked, then the loop body will be executed only once before program control drops to the next sequential program statement after the "while" loop. (True or False)

23N

24 That's right. The loop body will be skipped entirely.

24 B:320

25 Wrong. The loop body will be skipped entirely.

25 B:320

21Frame 320 T

22 *** For Loop ***

22

22 The "for" loop is a three part control structure. The first part is the loop control initialize expression, the second part is the loop control test expression, and the third part is the loop control increment expression.

22

22 The loop control initialize expression is an expression that is evaluated once and can serve to initialize variables used within the loop body. The loop control test expression is tested at the beginning of the loop and after execution of the body. Again, the loop control expression is "true" whenever it is "non zero" and "false" when it is "zero". Execution of the body will continue until the control expression is "false". If the expression is "false" the first time, then program control will drop to the next sequential program statement. The loop control increment expression is evaluated after execution of the loop body.

23B:325

21Frame 325 T

22 * For Loop Continued *

22

22 The structure of the "for" loop looks like this:

22

```

22  for (initialize_exp; test_exp; increment_exp)
22      statement_to_be_executed;
22

```

22

22 Again, the braces can be used to define a "block" of statements.

22 Such as:

22

```

22   for (loop_var = 0; loop_var < 50; loop_var++) {
22       first_statement;
22       next_statement;
22       last_statement;
22   }

```

23B:330

21Frame 330 T

```

22   * For Loop Continued *

```

22 Here is an example using the "for" loop control statement:

```

22   for (i = 0; i < 20; i++)
22       if((i % 2) == 0)
22           printf("i value is even");
22       else
22           printf("i value is odd");

```

22 When the "for" is encountered, the loop control initialize expression is executed setting i equal to zero. Next the loop control test expression is checked and found to be "true", so the loop body is then executed. After execution of the loop body the loop control increment expression is executed setting i equal to i plus one. The loop control test expression is then checked again. The execution of the loop body will continue until the loop control expression is no longer "true".

23B:335

21Frame 335 QM

22 Which of the following is "not" a part of the "for" loop control structure?

23A initialize expression

23

23B test expression

23

23C+ terminate expression

23

23D increment expression

24 Correct. Keep up the good work.

24 B:340

25ABD No. The loop structure has that as one of its parts.

25 B:340

25E I'm sorry, "E" was not one of your choices.

25 B:335

21Frame 340 T

```

22   * For Loop Continued *

```

22

22 As I mentioned before, loop structures serve basically the same purpose and can usually be accomplished by using one such structure. We have looked at both "while" and "for" loops so far. Let's compare the structure of these two loop types.

22

22 The "while" structure:

22

```

22   init_exp;

```

```

22   while (test_exp) {

```

22 The "for" structure:

22

```

22   for (init_exp; test_exp; incr_exp)

```

```

22       statement_to_be_executed;

```

```

22     statement_to_be_executed;    |
22     incr_exp;                    |
22 )                                |

```

22 Which of these two structures you use is up to you, but there are times
 22 when one may be more appropriate than the other.

23B:345

21Frame 345 T

22 *** Do-While Loop ***

22 The final loop structure available in C is the "do-while".

22 The "do-while" loop is a two part control structure just like the "do"
 22 loop. The basic difference between the "do" loop and the "do-while"
 22 loop is that the first part of the "do-while" is the executable body,
 22 and the second part is the loop control expression. This is just the
 22 opposite of the "do" loop control structure.

22 The loop body will be executed once, before the loop control expression
 22 is tested at the end of the loop. If the loop control expression is
 22 "true" then the loop body will be executed again. Execution of the loop
 22 body will continue until the loop control expression is "false". The
 22 biggest difference, I'm sure you have noticed, is that the loop body
 22 will be executed at least one time before program control drops to the
 22 next sequential program statement.

23B:350

21Frame 350 T

22 * Do-While Loop Continued *

22 The structure of the "do-while" loop looks like this:

```

22 do
22     statement_to_be_executed;
22 while (test_expression);

```

22 Of course, braces can be used to define a "block" of statements.
 22 It is suggested that you use braces at all times in order to avoid
 22 the confusion caused by the "while" statement at the end of the loop.
 22 It tends to look like the start of a "while" loop. The following the
 22 preferred format:

```

22 do {
22     statement_to_be_executed;
22 } while (test_expression);

```

23B:355

21Frame 355 QP

22The major difference between the "while" loop and the "do-while" loop is that
 22the "do-while" will always be executed at least once whereas the "while" loop
 22may be skipped altogether if the loop control expression is "false".

22(True or False)

23Y

24 Right.

24 B:360
 25 Wrong. That is a true statement.
 25 B:360
 21Frame 360 T
 22 *** Topic Review ***
 22
 22 In this topic we have looked at the "while", "for", and "do-while"
 22 loops.
 22
 22 We have seen many examples of what these statements and structures
 22 look like, and how they are used.
 22
 22 In the next topic area I will describe and show examples of the
 22 "break" and "continue" statements.
 22
 22 Hope I see you there!
 22
 22
 22 *** This concludes this topic area. ***
 23END
 31Frame 500 T BREAK AND CONTINUE STATEMENTS
 32 *** Break Statement ***
 32
 32 The "break" statement is used to terminate a "while", "for", or the
 32 "do-while" loop before the loop control expression becomes "false".
 32 It is also used in the "switch" control statement to prevent further
 32 statement execution after a "case" has been found that satisfies the
 32 switch.
 32
 32 When a "break" statement is encountered, it is executed and the loop
 32 or case in which it is located is terminated immediately. Program
 32 control then passes to the next sequential statement following the
 32 loop or switch.
 32
 32 I will show you how this looks in each of the loop structures as well
 32 as the "switch" structure.
 32
 32 But first I want to be sure you want to see these examples.
 33B:505
 31Frame 505 QP
 32Do you want to see examples of how the "break" statement is used?
 32(Yes or No)
 33Y
 34 OK. Here we go.
 34 B:510
 35 OK. Let's take a look at the "Continue" statement.
 35 B:535
 31Frame 510 T
 32 *** Break Statement Example #1: "While" loop ***
 32
 32 The following is an example of how the "break" statement can be used in
 32 the "while" loop.


```

32
32  exit = 0;                                ; Without getting into details of how the
32  while (exit == 0) {                      ; "scanf" statement works or where you
32      scanf("%d", &in_int);                ; would use this section of code, this
32      if(in_int < 0)                        ; example shows how the "break" statement
32          break;                            ; can be used to terminate the "while"
32      else                                  ; loop before the loop control statement
32          sum += in_int;                    ; becomes "false". If the variable named
32      if(sum > 100)                          ; "in_int" ever becomes a negative number,
32          exit++;                            ; the "break" will be executed and program
32  }                                          ; execution will continue with the next
32  next_sequential_statement;              ; sequential statement after the loop.

```

33B:515

31Frame 515 T

32 *** Break Statement Example #2: "For" loop ***

32
32 The following is an example of how the "break" statement can be used in
32 the "for" loop.

```

32
32  for (i=0; i<=10; i++) {
32      in_char = getchar();
32      if(in_char == '.')
32          break;
32      last_name[i] = in_char;
32  }
32  next_sequential_statement;

```

32 Here, the loop will be terminated if "in_char" becomes a period (.) and
32 program execution will, once again, continue with the next sequential
32 statement after the loop. Note: We will cover "arrays" in lesson 4
32 and "input & output" in lesson 6.

33B:520

31Frame 520 T

32 *** Break Statement Example #3: "Do-While" loop ***

32
32 The following is an example of how the "break" statement can be used in
32 the "do-while" loop.

```

32
32  count = 0;                                ; In this example, there are really
32  do {                                      ; two loop control expressions. The
32      count++;                              ; loop would be terminated if the
32      if(count > 10)                        ; value of "count" becomes greater
32          break;                            ; than 10, or if variable "avg_num"
32      avg_num = (total / tot_num);          ; ever exceeds the value of 69.
32      scanf("%d", &in_int);                ; Since the "do-while" is executed
32      tot_num++;                            ; before the loop control expression
32      total += in_int;                      ; is tested, the "break" statement
32  } while (avg_num < 70);                   ; could be used to control the loop's
32  next_sequential_statement;               ; execution the first time through.

```

33B:525

31Frame 525 T

32 *** Break Statement Example #4: "Switch" statement ***

32 The following is an example of how the "break" statement is used in
 32 the "switch" statement.

```

32 switch(temp) {           | The "break" is used in the "switch" statement
32     case 70:              | in order to prevent the unnecessary evalua-
32     case 80:              | tion of expressions that will turn out to be
32         nice++;           | "false". The "break" statement will termin-
32         break;            | ate the "switch" after the "case" is found
32     case 90:              | that is "true" and the statement(s) is execu-
32         hot++;            | ted. It is important to note that the execu-
32         break;            | tion of the switch is sequential, therefore
32     case 50:              | if in our example the value of "temp" is 70,
32         cool++;           | there is no need to check any "cases" after
32         break;            | the execution of the statement "nice++;"
32     };                    |
  
```

33B:530

31Frame 530 QM

32Which of the following will the "break" statement "not" work with?

33A "while" loop

33

33B "for" loop

33

33C "switch" statement

33

33D+ "if-else" statement

33

33E "do-while" loop

34 Right you are. The "if-else" works the same as the "switch" without the use
 34 of the "break" statement.

34 B:535

35ABCE Wrong. The "if-else" works the same as the "switch" without the use
 35 of the "break" statement.

35 B:535

31Frame 535 T

32 *** Continue Statement ***

32

32 The "continue" statement is used within a loop structure in order to
 32 force the loop's next iteration. The "continue" is used with the
 32 "while", "for", and "do-while" loops, but NOT with the "switch" state-
 32 ment.

32

32 When you use the "continue" in the "while" and "do-while" loops, it
 32 forces the immediate evaluation of the "loop control expression".

32

32 When you use the "continue" in the "for" loop, it executes the "loop
 32 control increment expression" and then the "loop control expression"
 32 is evaluated.

32

32 Let's take a look at an example.

33B:540

31Frame 540 T

32 * Continue Statement Continued *

32

32 The following is an example of the use of the "continue" statement in
32 a "for" loop.

32

```
32 for (i=0; i<max_i; i++) {  
32     if(name_area[i] != ' ')  
32         continue;  
32     num_found++;  
32 }
```

32

32 In this example the "continue" statement causes the loop to be executed
32 until a "space" is encountered or the "loop control test expression"
32 becomes "false". Once a space is found, "num_found" is incremented,
32 the "loop control increment expression" is executed, and the "loop
32 control test expression" is evaluated. Execution will continue in
32 this fashion until the "loop control test expression" becomes "false".

33B:545

31Frame 545 QP

32The use of the "continue" is only effective in the loop control structures
32of "while", "for", and "do-while". (True or False)

33Y

34 Right. It can be used in a "switch", but only if the switch is inside of
34 a loop structure, in which case it would cause the next iteration of the
34 loop structure.

34 B:550

35 Wrong. It can be used in a "switch", but only if the switch is inside of
35 a loop structure, in which case it would cause the next iteration of the
35 loop structure.

35 B:550

31Frame 550 T

32 *** Topic Review ***

32

32 In this topic we have looked at the "break" and "continue" statements.

32

32 I have presented you the opportunity to see many examples of how the
32 "break" statement is used in the three different loop structures as
32 well as the "switch" statement. You also saw an example of how the
32 "continue" statement can be used within a "loop" structure.

32

32 In the next topic area I will describe and show examples of the
32 "goto" and "label" statements.

32

32 Hope I see you there!

32

32

32

*** This concludes this topic area. ***

33END

41Frame 700 T GOTO STATEMENT AND LABELS

42 *** Introduction ***

42

42 The use of "goto" statements has come under attack within the software

42 engineering community of experts. Although most languages provide for
42 the use of "goto", it is highly discouraged. Most instances of the
42 statement can be eliminated by careful software development. This is
42 especially true in a language such as C.

42 Even though use of the statement is discouraged, it is a part of the
42 language and therefore I will give a brief description of how it is
42 used.

43B:705

41Frame 705 T

42 *** Label Statement ***

42 In order to use the "goto" statement, you must have some way of identi-
42 fying where to "goto" to. In other languages such as BASIC or Fortran,
42 this is done by using statement numbers. C doesn't use statement num-
42 bers but instead uses "labels".

42 A label is declared in a function by using the following form:

42 label_name:

42 When naming a "label", follow the same rules that you use when naming
42 a variable.

43B:710

41Frame 710 T

42 *** Goto Statement ***

42 The "goto" statement is used to transfer program control to some point
42 within a function other than the next sequential statement. The point
42 MUST be a labeled point in the same function.

42 The most common use of the "goto" statement is to terminate execution
42 of a deep nested loop structure. As we learned in the last topic area,
42 we can use the "break" statement to terminate a loop but it will only
42 terminate the inner most loop (the one it is physically in).

42 A "goto" statement has the following form: goto label_name;

42 Note: "goto" is one word. The use of: go to label_name; will cause
42 a compile error.

43B:715

41Frame 715 T

42 *** Goto/Label Example ***

42 The following two sections of code provide an example of how the "goto"
42 statement is used in conjunction with a label and how to write the same
42 section without using a "goto" statement.

42 Code with the "goto" & "label" ; Code without using the "goto"
42 -----
42 in_out() { ; in_out() {

```

42      char c;           |      char c;
42      begin:           |      do {
42          c = getchar(); |          c = getchar();
42          if(c!='\n') {  |          if(c!='\n')
42              printf("%c",c); |              printf("%c",c);
42              goto begin; } |          } while (c!='\n');
42      return;           |      return;
42  }                     |  }

```

43B:720

41Frame 720 QP

42You can only use the "goto" statement to transfer program control to a label
42within the function where the "goto" is located. (True or False)

43Y

44 Right.

44 B:725

45 Wrong. You can not transfer control to any other part of the program using
45 the "goto" statement.

45 B:725

41Frame 725 T

42 *** Lesson Three Summary ***

42

42 Well, we have come to the end of another lesson. If you have seen the
42 four subject topics in this lesson, you should now be ready to take
42 the final test. If you feel that you don't understand something well
42 enough to pass the test, please retake the topic that is giving you
42 problems.

42

42 Topic 1 described the "if", "if-else", "nesting", and "switch".

42

42 Topic 2 described the "while", "for", and "do-while" loops.

42

42 Topic 3 described the "break" and "continue" statements.

42

42 Topic 4 described the "label" and "goto" statements.

42

42 Good Luck on the test.

43END

51Frame 900 TT TEST OVER LESSON 3

52 Welcome to the final test of lesson three. This test consists of ten
52 questions over material presented in the previous four topic areas.

52

52 In order to successfully complete this lesson, you must achieve a
52 minimum score of 70% (seven out of ten questions correct).

52

52 If you miss a question, the correct answer will not be shown. It is
52 up to you to research the correct answer.

52

52 Well, enough said. Let's get on with it. Good luck!

53B:905

51Frame 905 QM

521. Which one of the following is "not" one of the control statements that
52was covered in this lesson?

53A if
 53
 53B if-else
 53
 53C switch
 53
 53D+ while
 54 Right. (1,100)
 54 B:910
 55ABC Wrong. (1,100)
 55 B:910
 55E "E" was not one of your choices.
 55 B:905
 51Frame 910 QP
 522. Braces "{}" are used to form a "block" of one or more statements to be
 52conditionally executed. (True or False)
 53Y
 54 Right. (1,120)
 54 B:915
 55 Wrong. (1,120)
 55 B:915
 51Frame 915 QM
 523. Since the "else" part of the "if-else" control structure is optional,
 52care must be taken to prevent which of the following from occurring?
 53A having the "else" statement skipped.
 53
 53B+ having the "else" applied to the wrong "if" statement.
 53
 53C having an "else" applied to two "if" statements.
 53
 53D having the "if" statement executed before the "else".
 54 Right. (1,150)
 54 B:920
 55BCD Wrong. (1,150)
 55 B:920
 55E "E" was not one of your choices.
 55 B:915
 51Frame 920 QP
 524. Essentially, the "switch" is just a special case of the "if-else"
 52structure, and its use is really just "programmer preference".
 52(True or False)
 53Y
 54 Right. (2,320)
 54 B:925
 55 Wrong. (2,320)
 55 B:925
 51Frame 925 QM
 525. If the "loop control expression" in the "while" loop is "false" the first
 52time it is checked, which of the following statements would be true?
 53A The loop body would be executed one time only.
 53
 53B The loop would be executed until the control expression becomes "true".

53
 53C+ The loop body would be skipped altogether.
 53
 53D The loop would become an infinite loop.
 54 Right. (2,305)
 54 B:930
 55ABD Wrong. (2,305)
 55 B:930
 55E "E" was not one of your choices.
 55 B:925
 51Frame 930 QM
 526. Which of the following is "not" a part of the "for" loop control
 52structure?
 53A initialize expression
 53
 53B test expression
 53
 53C increment expression
 53
 53D+ terminate expression
 54 Right. (2,320)
 54 B:935
 55ABC Wrong. (2,320)
 55 B:935
 55E "E" was not one of your choices.
 55 B:930
 51Frame 935 QP
 527. The major difference between the "while" loop and the "do-while" loop is
 52that the "do-while" will always be executed at least once whereas the "while"
 52loop may be skipped altogether if the loop control expression is "false".
 52(True or False)
 53Y
 54 Right. (2,345)
 54 B:940
 55 Wrong. (2,345)
 55 B:940
 51Frame 940 QP
 528. The "break" statement can only be used to terminate a "while" or "for"
 52loop before the "loop control expression" becomes false. (True or False)
 53N
 54 Right. (3,500)
 54 B:945
 55 Wrong. (3,500)
 55 B:945
 51Frame 945 QM
 529. Which of the following structures is the "continue" statement not
 52effectively used with?
 53A for loop
 53
 53B do-while loop
 53
 53C+ switch

53
53D while loop
54 Right. (3,535)
54 B:950
55ABD Wrong. (3,535)
55 B:950
55E "E" was not one of your choices.
55 B:945
51Frame 950 QM
5210. When using the "goto" statement in your C program, which of the
52following must be adhered to?
53A+ The target "label statement" must be in the same function.
53
53B The target "statement number" must be in the same function.
53
53C The "goto" statement must not be in a loop structure.
53
53D The "goto" statement must be before the "flagged" statement.
54 Right. (4,710)
54 B:955
55BCD Wrong. (4,710)
55 B:955
55E "E" was not one of your choices.
55 B:950
51Frame 955 T
52 *** End of Lesson Material ***
52
52 This marks the end of lesson number three. I hope that it was of some
52 benefit to you. I am looking forward to seeing you in lesson number
52 four. I hope that you didn't have too much trouble with the material
52 presented in this lesson. If you did, please voice your comments to
52 your training monitor who will in turn contact the CAI Plans Branch
52 at Keesler AFB, MS.
52
52 Well, let's take a look at how you did with the test ...
53END

File "LESSON4"

```
#      WW      WW  EEEEEEEE  LL      CCCCCC  000000  MMM  MMM  EEEEEEEE
#      WW WW WW  EE      LL      CC      00      00  MMM  MMM  EE
#      WW WW WW  EEEEE  LL      CC      00      00  MM  MM  MM  EEEEE
#      WWW  WWW  EE      LL      CC      00      00  MM  MM  MM  EE
#      WWW  WWW  EEEEEEEE  LLLLLLLL  CCCCCC  000000  MM      MM  EEEEEEEE
#
#
#
#      TTTTTTTTTT  00000000
#      TT      00      00
#      TT      00      00
#      TT      00      00
#      TT      00000000
#
#
#      LL      EEEEEEEE  SSSSSS  SSSSSS  000000  NN      NN      444
#      LL      EEEEEEEE  SSS  SSS  SSS  SSS  00000000  NNN  NN      4444
#      LL      EE      SSS      SSS      00      00  NNNN  NN      44 44
#      LL      EEEEE  SSSS      SSSS      00      00  NN  NN  NN      44 44
#      LL      EEEEE      SSSS      SSSS      00      00  NN  NN  NN      44444444
#      LL      EE      SSS      SSS      00      00  NN  NNNN      44444444
#      LLLLLLLL  EEEEEEEE  SSS  SSS  SSS  SSS  00000000  NN      NNN      44
#      LLLLLLLL  EEEEEEEE  SSSSSS  SSSSSS  000000  NN      NN      44
```

THE LESSON YOU ARE ABOUT TO TAKE CONTAINS INFORMATION ON ARRAYS, POINTERS,
AND ADDRESS ARITHMETIC USED IN C PROGRAMMING.

THE LESSON CURRENTLY CONSISTS OF FIVE TOPICS.

The Lesson Breakdown Is As Follows:

Topic 1: Introducing Arrays - This topic introduces the declaration,
initialization, and use of arrays. (Approx. time = 15 min.)

Topic 2: Introducing Pointers - This topic introduces the declaration
and use of pointers. (Approx. time = 15 min.)

Topic 3: Working with Pointers I - This topic is the first of two that
covers how to work with pointers. Emphasis is on how pointers
are passed to functions. (Approx. time = 10 min.)

Lesson Breakdown Continued:

Topic 4: Working with Pointers II - This topic is the second of two that
covers how to work with pointers. Emphasis is on how pointers
are used in conjunction with arrays and the use of address
arithmetic. (Approx. time = 10 min.)

```
#
# Topic 5: Lesson 4 Test - This is the lesson test over items that have
# been presented in the previous four lesson topics.
# (Approx. time = 5 min.)
#
```

```
# TOTAL LESSON TIME IS APPROXIMATELY 55 MINUTES.
#
```

```
# I hope that you enjoy it!
!
```

```
*****
*
*          SELECT THE TOPIC YOU WISH TO TAKE FROM THE FOLLOWING:
*
*****
```

STATUS	TOPIC #	TOPIC TITLE
@	1	Introducing Arrays
@	2	Introducing Pointers
@	3	Working with Pointers I
@	4	Working with Pointers II
@	5	Test Over Lesson 4

```
*****
* NOTE: A "STATUS" OF "+" INDICATES TOPIC SUCCESSFULLY COMPLETED.
*****
```

```
11Frame 100 T Introducing Arrays
```

```
12 *** Introduction ***
```

```
12
```

```
12 An "array" is a group of contiguously stored related variables.
```

```
12
```

```
12 In this topic area we will take a look at the basic use of arrays and
12 some advanced concepts involving arrays.
```

```
12
```

```
12
```

```
12 To be more specific, we will be looking at: one dimensional arrays,
12 multidimensional arrays, and array initialization.
```

```
12
```

```
12 Let's get started.
```

```
13B:105
```

```
11Frame 105 T
```

```
12 *** One Dimensional Arrays ***
```

```
12
```

```
12 The language C does not have a "string" variable type, therefore it
12 uses an array of characters to accomplish the same thing. If you
```

12 think of a string of characters such as a sentence. How would you
12 store it in your program? Well, the answer of course is to use an
12 array of characters.
12
12 The structure of the basic one dimensional character array declaration
12 statement is:
12
12 char var_name[n]; where "n" is the number of characters in the array.
12
12 Now comes the tricky part. The individual characters in the array
12 are called the "elements" of the array. Accessing these elements is
12 a very common procedure in programming. Let's look at an example that
12 uses an array and see how this is done.

13B:110

11Frame 110 T

12 * One Dimensional Arrays Continued *

12
12 For our example, let's say we want to store the word Payment . The
12 first thing we must do is decide on the size of the array that will
12 hold this word. This can be done by counting the number of characters
12 in the word. So, let's see... I count 8.

12
12 At first glance it looks like I made a mistake in counting the charac-
12 ters in Payment . This is not the case. In C the first element of
12 an array is stored in array position 0 (zero), and the last (string)
12 array position element is always a null character (\0). So, using
12 the following statement to declare our word as a character string
12 constant...

12
12 char ex_word[8] = "Payment";

12
12 the array will be filled as follows:

13B:115

11Frame 115 T

12 * One Dimensional Arrays Continued *

12
12 ex_word[0] = P
12 ex_word[1] = a
12 ex_word[2] = y
12 ex_word[3] = m
12 ex_word[4] = e
12 ex_word[5] = n
12 ex_word[6] = t
12 ex_word[7] = \0

12
12 The "null character" stored at the end of a string array is put there
12 automatically by the C compiler. All you have to worry about is to
12 leave room for it in your array. What if you don't want to worry
12 about such things? Well, there is a way to get around counting the
12 number of characters in a string constant and then adding one for the
12 null character. Let's take a look.

13B:120

11Frame 120 T

12 * One Dimensional Arrays Continued *

12

12 Using the statement: `char ex_word[] = "Payment";` will accomplish the same thing as the example we just looked at. Namely, an array consisting of eight elements will be declared and filled by the compiler.

12

12 The way in which the individual elements in an array are accessed is by referencing the element using an index. In our example an index of 4 would look like this: `ex_word[4]` and yield the character `e`.

12

12 Our discussion thus far has only dealt with the C character type. The use of arrays is by no means restricted to this C variable type. Here are a couple examples of arrays of other variable types:

12

12 `int ex_ints[35];` This is an array of integers (36 of them).

12

12 `float ex_floats[67];` An array of floating point reals (68 of them).

13B:125

11Frame 125 QM

12 Given the character array declaration: `char example[n] = "Example";`

12 Which of the following is the correct number for "n" ?

13A 10

13

13B 9

13

13C 8

13

13D 7

14 Right. Seven characters plus the "null character", therefore 8.

14 B:130

15ABD Wrong. There are seven characters plus the "null character",

15 `example[0]` thru `example[7]`, therefore the correct answer is 8 ("C").

15 B:130

11Frame 130 T

12 *** Multidimensional Arrays ***

12

12 As we have seen, a one dimensional array is declared using a statement such as `char ex_word[8];`. The dimension of this array is seen as a list of characters running from `ex_word[0]` to `ex_word[7]`.

12

12 A two dimensional array can be thought of as a table consisting of rows and columns. The way in which a two dimensional array is declared is as follows:

12

12 `int ex_int[n][m];` where "n" is the number of rows
12 and "m" is the number of columns.

12

12 Let's look at an example.

13B:135

11Frame 135 T

12 * Multidimensional Arrays Continued *

```

12
12   If we want to store the test scores for a class of 5 students who have
12   each taken 4 tests, we could do it like this:
12
12   int scores[5][4] = {
12       {75,80,70,95},
12       {85,85,90,95},
12       {60,90,80,90},
12       {70,80,90,90},
12       {75,85,95,85}
12   };
12
12   This form is very representative of how the table would look. How these
12   numbers are stored is as follows: scores[0][0] = 75, scores[0][1] = 80,
12   scores[0][2] = 70, and scores[0][3] = 95. You then increment the first
12   index and continue: scores[1][0] = 85 ... scores[4][3] = 85.
138:140
11Frame 140 T
12   * Multidimensional Arrays Continued *
12
12   In our example, we defined an array with 5 rows and 4 columns. We
12   also filled the array with test scores. Of course these test scores
12   are useless unless we have defined the student that each row represents.
12   This can be done several ways, but I would define a symbolic constant
12   for each student. Such as: #define Jones 0
12                               #define Smith 1
12                               #define Brown 2
12                               #define Green 3
12                               #define White 4
12
12   Now if you want to find out what Brown got on his third test you could
12   use the statement: Brown_3 = scores[Brown,2]; This will retrieve the
12   score stored in array position scores[2][2], which was 80.
12
12   A good way I've found to get used to arrays is to experiment with them.
138:145
11Frame 145 T
12   * Multidimensional Arrays Continued *
12
12   As you might have deduced by now, you can define arrays of more than
12   two dimensions. All that needs to be done is add more brackets ([])
12   after the array name.
12
12   For example: int four_D_array[5][10][5][20];
12
12   Don't ask me to give you a visual picture of such a thing, but I can
12   tell you that there are 5000 integer storage locations allocated by
12   such a declaration (5 x 10 x 5 x 20 = 5000).
138:150
11Frame 150 QM
12Given the array declaration: int array[2][5] = {
12                               {75,80,70,95,65},

```

```

12                                     (85,60,90,50,55) );
12
12Which of the following is the value stored in position array[1][2] ?
13A 80
13B 60
13C+ 90
13D 85
13E 70
14 Very good.
14 B:155
15ABDE No. Answer "C" is the correct one.
15 B:155
11Frame 155 T
12 *** Array Initialization ***
12
12 We have already seen some of the ways in which arrays are initialized.
12 When I gave an example of a one dimensional character array I used the
12 statement:
12
12     char ex_word[8] = "Payment";
12
12 That is one way to initialize the character array, another way would be:
12
12     char ex_word[] = "Payment";
12
12 Yet another way would be:
12
12     char ex_word[] = {'P','a','y','m','e','n','t','\0'};
12
12 All the above are correct if the array is a "global" array.
13B:160
11Frame 160 T
12 * Array Initialization Continued *
12
12 You may NOT initialize arrays that are "automatic". This means any
12 arrays that are contained within a function. In order to initialize
12 an array within a function it must be declared as "static". The way
12 this is done is by use of the keyword "static".
12
12 For example:
12
12 This initialization is wrong.      | This is the correct way.
12 -----
12 sample() {                          | sample() {
12     char array[] = "Example";        |     static char array[] = "Example";
12     .                                |     .
12     .                                |     .
12     .                                |     .
12 }                                    | }
13B:165
11Frame 165 T
12 * Array Initialization Continued *

```

```

12
12 When initializing arrays other than character arrays, the initializing
12 is accomplished with values enclosed in braces. For example:
12
12 A one dimensional global integer array can be initialized using:
12
12     int array[5] = {24,67,82,90,41};
12
12 Or, if all values of the array are being specified, the dimension can
12 be left out, as in:
12
12     int array[] = {24,67,82,90,41};
12
12 Again, if the array is local to a function and needs to be initialized,
12 use the keyword "static".
13B:170
11Frame 170 T
12 * Array Initialization Continued *
12
12 Multidimensional arrays are initialized by rows, as in one of our pre-
12 vious examples:
12
12     int scores[5][4] = {      |      int scores[][] = {
12         {75,80,70,95},        |         {75,80,70,95},
12         {85,85,90,95},        |         {85,85,90,95},
12         {60,90,80,90},        |         {60,90,80,90},
12         {70,80,90,90},        |         {70,80,90,90},
12         {75,85,95,85}         |         {75,85,95,85}
12     };                        |      };
12
12 If any of the values are missing, then the array value will be stored
12 as 0 (zero). Note: If values are missing, than dimensions must be
12 specified. Of course "static" must be used for local function arrays
12 that you want to initialize.
13B:175
11Frame 175 QP
12The integer array initialization: int array[] = {2,4,6,8}; is valid for
12a one dimensional integer array having 5 elements. (True or False)
13N
14 Right. If you intend for the array to have 5 elements then either 5 values
14 must be give in the list or a dimension of 5 must be explicitly stated.
14 B:180
15 Wrong. If you intend for the array to have 5 elements then either 5 values
15 must be give in the list or a dimension of 5 must be explicitly stated.
15 B:180
11Frame 180 T
12 *** Topic Review ***
12
12 In this topic we have looked at "one dimensional" and "multidimensional"
12 arrays. We have also seen how to initialize these arrays.
12
12 We have seen examples of what these arrays and initialization statements

```

```

12 look like, and how they are used.
12
12 In the next topic area I will describe pointers and give a few examples
12 of their use.
12
12 See you there!
12
12
12 *** This concludes this topic area. ***
13END
21Frame 300 T Introducing Pointers
22 *** Introduction ***
22
22 A "pointer" is a variable that contains the address of where some other
22 variable resides in memory.
22
22 In this topic area I will describe how pointers are declared and used
22 within a C program.
22
22 Since pointers can be very confusing to someone who has not seen them
22 before, I will restrict my discussion to elementary concepts and leave
22 their more advanced uses for your research.
22
22 Let's get started.
23B:305
21Frame 305 T
22 *** Pointers ***
22
22 In the declaration: int var_one = 500; a storage location is set
22 aside in memory for an integer variable and the value of 500 is
22 stored in that memory location. That memory location also has a
22 memory address.
22
22 In C, you can determine the memory address by the use of the unary
22 operator & .
22
22 The way that you would assign a pointer variable to the memory location
22 where "var_one" is located is: point_v1 = &var_one; this assigns the
22 address of the variable "var_one" to the variable "point_v1".
22
22 Note: Pointer names follow the same rules as other variable types and
22 must be declared as the same type of the variable being pointed
22 to (as we'll see later).
23B:310
21Frame 310 T
22 * Pointers Continued *
22
22 That's fine. Now we know how to find out the memory address of a
22 variable, but what good is it?
22
22 It would be nice if we could now find out the value stored at the
22 address pointed to by our pointer. C just happens to have a special

```


22 operator that allows us to do just that.
22
22 In C, you can determine the value stored at an address pointed to by
22 pointer by the use of the unary operator * .
22
22 The way that you would use this operator to find the value stored at
22 a pointed to address is: var1_val = *point_v1; this statement assigns
22 the value stored at the memory location pointed to by "point_v1" to
22 the variable "var1_val". Which, in our example, would be 500.

23B:315

21Frame 315 T

22 * Pointers Continued *

22
22 To help clear up what we have done so far, let's look at our example
22 again and compare it to statements we are familiar with.

22
22 The sequence of statements: var_one = 500;
22 point_v1 = &var_one;
22 var1_val = *point_v1;

22 Is the same as the sequence of statements: var_one = 500;
22 var1_val = var_one;

22
22 In both of the above cases, the variable "var1_val" is assigned the
22 value of 500. Although the use of the first set of statements seems
22 to be an unnecessary complication of a straightforward assignment,
22 keep in mind that this is just an example to demonstrate how a pointer
22 is used but does not show the true power of pointer usage.

23B:320

21Frame 320 QM

22The two unary operators used when working with pointers are the ____ and
22the ____.

23A # and &

23

23B+ & and *

23

23C \$ and &

23

23D \$ and #

23

23E # and *

24 Right.

24 B:325

25ACDE Wrong. Answer "B" is the correct response.

25 B:325

21Frame 325 T

22 *** Pointer Declaration ***

22

22 In order for pointers to be used in a C program, you must declare a
22 pointer variable before you can use it. The type of the pointer
22 variable must be the same as the variable that it is to point to.

22

22 In our example, the statement: `point_v1 = &var_one;` must be preceded
 22 by the declaration: `int *point_v1;` which states that the value to be
 22 pointed to by "point_v1" is of type "int".
 22
 22 Pointers to other types of variables are declared in the same way.
 22 For example:
 22
 22 `char *char_point;` declares the pointer variable "char_point" which
 22 is to point to a variable of type "char".
 23B:330
 21Frame 330 QP
 22The declaration: `float *var_point;` declares the pointer variable
 22 "var_point" to be of type "float".
 23N
 24 Very good. It declares the pointer variable "var_point" which "points"
 24 to a variable of type "float".
 24 B:335
 25 No. It declares the pointer variable "var_point" which will "point"
 25 to a variable of type "float".
 25 B:335
 21Frame 335 T
 22 *** Pointer Facts ***
 22
 22 Pointers can be used in expressions. For example:
 22
 22 `answer = *point + 35;` adds 35 to the value pointed to by "point" and
 22 stores the result in variable "answer".
 22
 22 `*p_1 = *p_2 * 5;` multiplies the value pointed to by "p_2" by 5 and
 22 stores the result in the variable pointed to by "p_1".
 22
 22 `p_one = p_two;` will make "p_one" point to the same variable that
 22 "p_two" points to if both "p_one" and "p_two" are
 22 declared to point to the same variable type.
 22 (i.e. `int *p_one, *p_two;`)
 23B:340
 21Frame 340 T
 22 *** Topic Review ***
 22
 22 In this topic we have looked at pointer declaration and a few elemen-
 22 tary examples of how they are used.
 22
 22 The rest of this lesson will discuss some other uses of pointers in
 22 C programming.
 22
 22 In the next topic area (3) I will describe and show examples of how
 22 to pass pointers as function arguments. In topic area four I will
 22 discuss the use of pointers in conjunction with arrays and explain
 22 how to do address arithmetic.
 22
 22 Hope I see you there!
 22

```

22
22          *** This concludes this topic area. ***
23END
31Frame 500 T Working with Pointers I
32  *** Introduction ***
32
32  In the last topic area we saw that a "pointer" is actually a vari-
32  able that contains the address of where some other variable resides
32  in memory.
32
32  In this topic area I will describe how pointers are passed to func-
32  tions, a rationale for doing it, and a few examples.
32
32  Let's get started!
33B:505
31Frame 505 T
32  *** Function Augument Background ***
32
32  We have seen two methods of passing arguments to a function, although
32  I have not explicitly named these methods. Now is as good a time as
32  any to do so. They are: "Call by value" and "Call by reference".
32  The main difference in the two is that the actual value stored in a
32  variable can only be changed by using the "Call by reference" method
32  of argument passing. Let's look at a couple of examples to help make
32  this clear.
32
32  Let's say we have a C program that has two functions "main" and "add".
32  The "main" function calls the "add" function and passes it two vari-
32  ables "x" and "y". The "add" function takes the two arguments and
32  adds 50 to the first (x) and 75 to the second (y). The "main" func-
32  tion then prints out the two variables "x" and "y".
32
32  Let's see what these two functions might look like.
33B:510
31Frame 510 T
32  add(x,y)          ; This is a clear example of the "Call by
32  int x,y;          ; value" method. Even though I called the
32  {                ; two variables the same name in both of
32  x += 50;          ; the functions, each function has its own
32  y += 75;          ; copy of the variables. Hence, the actual
32  return;           ; values of "x" and "y" in "main" are never
32  }                ; changed by the function "add". This will
32                  ; result in "10" and "30" being printed by
32                  ; the "main" function. One way around this
32  main() {          ; problem is to make "x" and "y" global to
32  int x,y;          ; both functions. The perfered method is to
32  x = 10;           ; use "pointers" as we will see shortly.
32  y = 30;           ; To introduce us to the concept used in
32  add(x,y);         ; passing pointers, let's look at another
32  printf("\n%d  %d",x,y); ; example.
32  }
33B:515

```

31Frame 515 T

```
32 For this example let's say we have two functions "main" and "init".
32 The "main" function declares an array called "line" to be a sequence
32 of 80 characters. The "main" function calls the "init" function and
32 passes it the array to be initialized to blanks.
32
32 init(b_line)           | This is a clear example of the "Call by ref-
32   char b_line[];       | erence" method. Although I called the array
32   {                     | different names in the two functions, the
32     for (i=0;i<80;i++) | "init" function will actually change the
32       b_line[i] = ' '; | array "line" declared in function "main".
32   }                     | This is because the function "main" actually
32                           | passes the address of where the array "line"
32 main() {                 | begins in memory to the function "init".
32   char line[80];         | This "Call by reference" only works in the
32   init(line);            | case of arrays. Before we look at pointer
32 }                         | passing, let me ask you a quick question.
```

33B:520

31Frame 520 QP

32The "Call by value" method of argument passing only passes a copy of a
32variable, whereas the "Call by reference" method passes the address of
32the argument. (True or False)

33Y

34 Right. You have been paying close attention.

34 B:525

35 Wrong. I hope you aren't falling asleep on me.

35 B:525

31Frame 525 T

32 *** Passing Pointers ***

32

32 We've seen in another lesson that a called function can only return
32 one value to the calling function. Thus, only one value of the call-
32 ing function is truly changed. This of course precludes the use of
32 global variables by the functions in question.

32

32 If it is necessary for the called function to change more than one
32 variable of the calling function, then the preferred method is to use
32 addresses or pointers as passed arguments.

32

32 There are three ways in which to accomplish the task introduced above.

32

32 1. Pass the address of the variable.

32

32 2. Pass a pointer to the variable.

32

32 3. Pass an array name.

32

33B:530

31Frame 530 T

32 * Passing Pointers Continued *

32

32 If we have a function that is to be called and its "function" is to

32

32 change two variables (as in our first example), we can set up the

32

32 function to receive pointers as its arguments as follows:

32

```

32  add(px,py)           | In this example I have identified the variables
32  int *px,*py;         | "px" and "py" to be pointers to variables of
32  {                    | type "int". When the function is executed, the
32  *px += 50;           | values stored in the variables, pointed to by
32  *py += 75;           | these pointers, will change by "50" and "75"
32  return;              | respectively.
32  }

```

32 Let's look at how we would pass the "addresses" of the variables to this
 32 function from our "main" function.

33B:535

31Frame 535 T

32 * Passing Pointers Continued *

32 One way we have identified as being a way to pass a pointer to a
 32 function is by passing the "address". The following illustrates
 32 this method.

```

32  main() {              | In this example the only statement that
32  int x,y;              | has changed from when you last saw it is
32  x = 10;               | the "add" function call statement. All
32  y = 30;               | I did was to use the unary operator &
32  add(&x,&y);            | to identify the arguments as the address
32  printf("\n%d %d",x,y); | of the variables.
32  }

```

32 Now let's look at another way to pass pointers from the calling
 32 function to the called function.

33B:540

31Frame 540 T

32 * Passing Pointers Continued *

32 An alternate way of passing pointer information is to pass the pointer
 32 itself. The following illustrates this method.

```

32  main() {              | In this example the variables "px" and
32  int x,y,*px,*py;      | "py" are identified as pointers to vari-
32  x = 10;               | ables of type "int". The addresses of
32  y = 30;               | the variables "x" and "y" are stored in
32  px = &x;              | those pointer variables and they are used
32  py = &y;              | as arguments in the "add" function call
32  add(px,py);           | statement. Again, after execution of the
32  printf("\n%d %d",x,y); | "add" function, the new values of "x" and
32  }                    | "y" will be printed out.

```

32 The third method of passing pointer information (pass an array name)
 32 was already discussed.

33B:545

31Frame 545 QM

32 Which of the following is "not" one of the ways in which to pass information
 32 that will allow the value of a variable to be changed by a called function?

33A Pass a pointer to the variable.

33
 33B Pass an array name.
 33
 33C+ Pass the variable name.
 33
 33D Pass the address of the variable.
 34 Very good.
 34 B:550
 35ABD No. That is one of the ways "to" do it. The correct response is "C".
 35 B:550
 35E "E" was not a given choice. Please try again.
 35 B:545
 31Frame 550 T
 32 *** Topic Review ***
 32
 32 In this topic area we have looked at the "Call by reference" and "Call
 32 by value" methods of argument passing as well as how to pass pointers
 32 as function arguments.
 32
 32 We have seen several examples to help illustrate all of these methods.
 32
 32 In the next topic area I will describe the use of pointers in conjunc-
 32 tion with arrays and explain how to use address arithmetic.
 32
 32 Hope to see you there!
 32
 32
 32 *** This concludes this topic area. ***
 33END
 41Frame 700 T Working with Pointers II
 42 *** Introduction ***
 42
 42 In this topic area I will describe how pointers are used in conjunction
 42 with arrays and how to use address arithmetic.
 42
 42 We have seen already that when you declare an array with a statement
 42 like: char line[] = "This is an example"; the compiler sets up 19
 42 contiguous storage locations in memory. These locations have names
 42 line[0] thru line[18].
 42
 42 We also have seen how to refer to each individual storage location
 42 using an "index" value. If "i" is a integer then line[i] refers to
 42 the "i"th element in array "line". You can manipulate "i" in order
 42 to give you quick and easy access to any of the elements of the array.
 42
 42 Let's now see how we can use pointers to give us access and manipula-
 42 tive power over arrays.
 43B:705
 41Frame 705 T
 42 *** Array Access Thru Pointers ***
 42
 42 When an array is declared (char line[10];) the array can be passed

42 between functions by just giving the array name. For example:
42
42 init(line); This calls the function "init" and passes the array "line".
42
42 What actually happens is the C compiler passes the address of the "0"th
42 element of the array. So in essence, a pointer to the beginning of the
42 array is passed ("line" being the pointer).

43B:710

41Frame 710 T

42 * Array Access Thru Pointers Continued *

42
42 The same thing can be accomplished by explicitly defining a pointer
42 in the following manner:

42
42 char *p_line; This identifies "p_line" as a pointer to a variable of
42 type "char".

42
42 p_line = &line[0]; This assigns the address of the "0"th element of
42 array "line" to the pointer variable "p_line".

42
42 init(p_line); This calls the function "init" and passes the address
42 of the starting location of array "line".

42
42 Once the above declarations have been made, the two expressions:
42 "line" and "p_line" are interchangeable.

43B:715

41Frame 715 QM

42If you have the declaration: char line[10]; which of the following state-
42ments will assign the address of the "0"th element to a pointer variable
42that has been declared using the statement: char *p_line; ?

43A *p_line = line[0];

43

43B p_line = line[0];

43

43C *p_line = &line[0];

43

43D+ p_line = &line[0];

44 Right.

44 B:720

45ABC Wrong. Answer "D" is the correct response.

45 B:720

45E "E" was not a given choice. Please try again.

45 B:715

41Frame 720 T

42 * Array Access Thru Pointers Continued *

42

42 The next logical step in our discussion is to look at how we can access
42 the individual elements of an array using our declared pointer.

42

42 We already know that "line[0]" will give us access to the "0"th element
42 of the array "line", but now that "p_line" has the address of the "0"th
42 element of the array, we can also use the expression "*p_line" to accom-

42 plish the same effect. Note: It is also legal to use the notation
42 "p_line[0]", but we will avoid this to cut down on the confusion.

42
42 Now that we have pointer access to the array, we can manipulate the
42 pointer to point to any of the array elements by use of address arith-
42 metic.

43B:725

41Frame 725 T

42 *** Address Arithmetic ***

42
42 The most common use of address arithmetic is through the use of the
42 increment, decrement, addition, and subtraction operators.

42
42 The operation must involve a pointer and an integer with the exception
42 of the subtraction operator (subtraction/comparison of two pointers is
42 allowed).

42
42 The use of "relational" operators is legal as long as the pointers
42 point to members of the same array. The use of the "operational
42 assignment" operators "+=" and "-=" are also legal.

42
42 Let's look at an example of how to use some of these operators.

43B:730

41Frame 730 T

42 * Address Arithmetic Continued *

42
42 When we first started this topic area I used the declaration state-
42 ment: `char line[] = "This is an example";` to declare and initialize
42 the array "line".

42
42 Using the declarations: `char *p_line;` and `p_line = &line[0];` we
42 established a pointer to the "0"th element of array "line".

42
42 We also saw that the expressions "line[0]" and "*p_line" are equivalent.

42
42 Both would return a value of T if used in a statement such as:

42
42 `char_val = line[0];` OR `char_val = *p_line;`

43B:735

41Frame 735 T

42 * Address Arithmetic Continued *

42
42 We can move forward and backward in the array by using our pointer
42 and the legal operators mentioned before.

42
42 If we want to move one element forward in the array we can use the in-
42 crement operator (++), the addition operator (+), or the operational
42 assignment operator (+=).

42
42 For example: `p_line++` will make the pointer point to the next se-
42 quential element in the array. Likewise, `p_line = p_line + 1;` and
42 `p_line += 1;` will have the same effect.

42
 42 In general, it can now be said that if "p_line" is a pointer and "i"
 42 is an integer, then p_line += i will increment "p_line" by "i" thus
 42 making "p_line" point to an element "i" elements beyond its present
 42 location. Decrementing is done in a similar fashion.
 43B:740
 41Frame 740 QM
 42Given that "pa_val", "pb_val", and pc_val are pointers. Which of the fol-
 42lowing statements is "not" a "legal" address arithmetic operation?
 43A+ pc_val = pb_val + pa_val;
 43
 43B pc_val = pb_val - pa_val;
 43
 43C pa_val += (pb_val += pc_val);
 43
 43D pa_val -= (pb_val - pc_val);
 44 Very good. Addition of two pointers is not allowed.
 44 B:745
 45BCD Wrong. That is a valid statement involving address arithmetic.
 45 B:745
 45E "E" was not a given choice. Please try again.
 45 B:740
 41Frame 745 T
 42 *** Lesson Four Summary ***
 42
 42 Well, we have come to the end of lesson four. If you have seen the
 42 four subject topics in this lesson, you should now be ready to take
 42 the final test. If you feel that you *don't understand something well*
 42 enough to pass the test, please retake the topic that is giving you
 42 problems.
 42
 42 Topic 1 gave an introduction to one and multidimensional arrays.
 42
 42 Topic 2 gave an introduction to pointers and their use.
 42
 42 Topic 3 gave a description of how pointers are passed to functions.
 42
 42 Topic 4 gave a description of pointer use in conjunction with arrays.
 42
 42 Good Luck on the test.
 43END
 51Frame 900 TT TEST OVER LESSON 4
 52 Welcome to the final test of lesson four. This test consists of ten
 52 questions over material presented in the previous four topic areas.
 52
 52 In order to successfully complete this lesson, you must achieve a
 52 minimum score of 70% (seven out of ten questions correct).
 52
 52 If you miss a question, the correct answer will not be shown. It is
 52 up to you to research the correct answer.
 52
 52 Well, enough said. Let's get on with it. Good luck!

53B:905

51Frame 905 QM

521. In the array declaration: char word[x] = "Sample"; which of the following is the correct value for "x" ?

53A 9

53

53B 8

53

53C+ 7

53

53D 6

54 Right. (1,110)

54 B:910

55ABD Wrong. (1,110)

55 B:910

55E "E" was not one of your choices.

55 B:905

51Frame 910 QM

522. Given the array declaration: int array[2][4][6]; how many integer storage locations are allocated?

53A 12

53

53B 24

53

53C 36

53

53D+ 48

54 Right. (1,145)

54 B:915

55ABC Wrong. (1,145)

55 B:915

55E "E" was not one of your choices.

55 B:910

51Frame 915 QP

523. The integer array initialization: int array[5] = {4,8,12}; is valid for a one dimensional integer array having 5 elements. (True or False)

53Y

54 Right. (1,170)

54 B:920

55 Wrong. (1,170)

55 B:920

51Frame 920 QM

524. Which of the following is the unary operator that is used to determine the memory address of a variable?

53A @

53

53B #

53

53C %

53

53D+ &

53

53E *

54 Right. (2,305)

54 B:925

55ABCE Wrong. (2,305)

55 B:925

51Frame 925 QP

525. The declaration: char *char_point; declares the pointer variable

52"char_point" which points to a variable of type "char". (True or False)

53Y

54 Right. (2,325)

54 B:930

55 Wrong. (2,325)

55 B:930

51Frame 930 QP

526. The "Call by reference" method of argument passing only passes a copy

52of a variable, whereas the "Call by value" method passes the address of

52the argument. (True or False)

53N

54 Right. (3,510-515)

54 B:935

55 Wrong. (3,510-515)

55 B:935

51Frame 935 QM

527. Given: main () {

52 int x,y,*px,*py;

52 x = y = 0;

52 px = &x;

52 py = &y;

52 change(px,py); }

52

52Which of the following is the method of pointer passing used?

53A+ Pass a pointer to the variable.

53B Pass an array name.

53C Pass the address of the variable.

53D Pass the variable name.

54 Right. (3,540)

54 B:940

55BCD Wrong. (3,540)

55 B:940

55E "E" was not one of your choices.

55 B:935

51Frame 940 QM

528. Given the declaration: int array[10]; which of the following state-

52ments will assign the address of the third element to a pointer variable

52that has been declared using the statement: int *p_array; ?

53A *p_array = &array[2];

53

53B+ p_array = &array[2];

53

53C *p_array = array[2];

53

53D p_array = array[2];

54 Right. (4,710)
 54 B:945
 55ACD Wrong. (4,710)
 55 B:945
 55E "E" was not one of your choices.
 55 B:940
 51Frame 945 QP
 529. The statement: `init(p_var);` calls the function "init" and passes the
 52address of the variable pointed to by the pointer "p_var", provided the
 52pointer was declared using a statement like "`int *p_var;`". (True or False)
 53Y
 54 Right. (4,710)
 54 B:950
 55 Wrong. (4,710)
 55 B:950
 51Frame 950 QM
 5210. Which of the following operators is "not" a legal operator in address
 52arithmetic?
 53A +
 53
 53B -
 53
 53C +=
 53
 53D --
 53
 53E+ /
 54 Right. (4,725)
 54 B:955
 55ABCD Wrong. (4,725)
 55 B:955
 51Frame 955 T
 52 *** End of Lesson Material ***
 52
 52 This marks the end of lesson number four. I hope that it was of some
 52 benefit to you. I am looking forward to seeing you in lesson number
 52 five. I hope that you didn't have too much trouble with the material
 52 presented in this lesson. If you did, please voice your comments to
 52 your training monitor who will in turn contact the CAI Plans Branch
 52 at Keesler AFB, MS.
 52
 52 Well, let's take a look at how you did with the test ...
 53END

File "LESSON5"

```
#      WW      WW  EEEEEEEE  LL      CCCCCC  000000  MMM  MMM  EEEEEEEE
#      WW WW WW  EE          LL      CC      00   00  MMM  MMM  EE
#      WW WW WW  EEEEE     LL      CC      00   00  MM MM MM  EEEEE
#      WWW  WWW  EE          LL      CC      00   00  MM MM MM  EE
#      WWW  WWW  EEEEEEEE  LLLLLLLL  CCCCCC  000000  MM      MM  EEEEEEEE
#
#
#
#      TTTTTTTTTT  00000000
#      TT      00      00
#      TT      00      00
#      TT      00      00
#      TT      00000000
#
#
#      LL      EEEEEEEE  SSSSSS  SSSSSS  000000  NN      NN  55555555
#      LL      EEEEEEEE  SSS SSS  SSS SSS  00000000  NNN      NN  55555555
#      LL      EE          SSS      SSS      00   00  NNNN      NN  55
#      LL      EEEEE     SSSS      SSSS      00   00  NN NN NN  555555
#      LL      EEEEE     SSSS      SSSS      00   00  NN NN NN  555555
#      LL      EE          SSS      SSS      00   00  NN  NNNN      555
#      LLLLLLLL  EEEEEEEE  SSS SSS  SSS SSS  00000000  NN      NNN  55555555
#      LLLLLLLL  EEEEEEEE  SSSSSS  SSSSSS  000000  NN      NN  555555
```

THE LESSON YOU ARE ABOUT TO TAKE CONTAINS INFORMATION ON STRUCTURES THAT
ARE USED IN C PROGRAMMING.

THE LESSON CURRENTLY CONSISTS OF FIVE TOPICS.

The Lesson Breakdown Is As Follows:

Topic 1: Introducing Structures - This topic introduces the idea of
structures and two methods of declaring them.
(Approx. time = 10 min.)

Topic 2: Structures and Arrays - This topic describes the use of struc-
tures within structures and arrays of structures.
(Approx. time = 5 min.)

Topic 3: Structures and Pointers - This topic describes how to use point-
ers in conjunction with structures. (Approx. time = 5 min.)

Lesson Breakdown Continued:

Topic 4: Structures and Functions - This topic describes how structures
are passed between functions. (Approx. time = 5 min.)

林林林林林

✱

*
*
*

中國

圖 3-1-1

家恒事

✱
✱
✱

1
1
1

1
1
.

1
1
1

1
1
1

1
1
1

1
1
1

1
1

11

1
1

1
1

1

1
1
1

1
1
1

11

1

1

```

12 Whenever you have a group of related items it is nice to be able to
12 group them in such a way as to give quick and easy access. In C,
12 the way this is done is through the use of "structures".
12
12 For example, if you have information about a student at a university,
12 this information might include items such as: Name, Address, Major,
12 GPA, and Advisor. Instead of keeping all this information stored
12 separately we can form a structure with five parts containing the
12 needed information.
12
12 Let's take a look at one way to declare our structure.
13B:110
11Frame 110 T
12 *** Declaring Structures ***
12
12 Our first way of declaring a structure uses the keyword "struct"
12 followed by an open brace "{" followed by the declaration of the
12 item variables followed by the close brace "}" followed by the
12 structure name followed by a semicolon.
12
12 For our example this would look something like this:
12
12 struct (                               ; Each of the character arrays must
12     char name[NAME_SIZE];              ; have predeclared constant values
12     char address[ADDRESS_SIZE];        ; for their sizes, hence the use of
12     char major[MAJOR_SIZE];            ; capital letter names. You could
12     float gpa;                         ; have broken "name" or "address"
12     char advisor[ADVISOR_SIZE];        ; into several variables or even
12 } student;                             ; other structures as we'll see later.
13B:115
11Frame 115 T
12 * Declaring Structures Continued *
12
12 The "structure name" need not be a single variable name. You can give
12 several different names to the same structure type by listing the names
12 seperated by commas.
12
12 For example:
12
12 struct (
12     int wing_span;
12     int num_tires;
12     double tonage;
12     double fuel_cap;
12 } F_16, C_141, C_5A, KC_135;
12
12 This example shows how you can define a standard information structure
12 that can be used for several different types of aircraft.
13B:120
11Frame 120 QP
12The use of structures allows for the grouping of related variables into
12a form which will be easy and quick to access. (True or False)

```

13Y

14 That's right.

14 B:125

15 Wrong. It is easy and quick, as you will shortly see.

15 B:125

11Frame 125 T

12 * Declaring Structures Continued *

12

12 Our second way of declaring a structure uses a sort of "template" for
12 the composition of the structure variables.

12

12 This way of declaring a structure uses the keyword "struct" followed by
12 a structure tag followed by an open brace "{" followed by the declara-
12 tion of the item variables followed by the close brace "}" followed by
12 a semicolon. For our "student" example this would look something like:

12

```
12 struct stu_rec {           | As you can see, the structure name
12     char name[NAME_SIZE];   | has been dropped and I have added the
12     char address[ADDRESS_SIZE]; | structure tag name of "stu_rec".
12     char major[MAJOR_SIZE];  | Defining structures in this way will
12     float gpa;              | allow you to define a variable of
12     char advisor[ADVISOR_SIZE]; | this type within your program when-
12 };                           | ever you need it.
```

13B:130

11Frame 130 T

12 * Declaring Structures Continued *

12

12 The major difference between the two methods of declaring structures
12 is that the first method will allocate memory space for the structure
12 variable when the program is run through the C compiler, and the second
12 method doesn't.

12

12 The second method only defines a structure type which you can use in
12 later variable declarations. For example, if you have several students
12 that you wish to identify within your program, you can use the follow-
12 ing declaration to allocate memory space for them:

12

```
12 struct stu_rec student_1, student_2, student_3;
```

12

12 This declares the variables "student_1", "student_2", and "student_3"
12 to be structures of type "stu_rec".

13B:135

11Frame 135 T

12 *** Structure Variable Access ***

12

12 Now that we have seen how to declare structures, it is now time to see
12 how to access the individual members of the structure.

12

12 Access to these individual structure members is gained through the use
12 of the structure member operator . (period).

12

AD-A163 842

COMPUTER ASSISTED INSTRUCTION FOR THE 'C' PROGRAMMING
LANGUAGE ON THE ZEN. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. F W DEMARCO

3/3

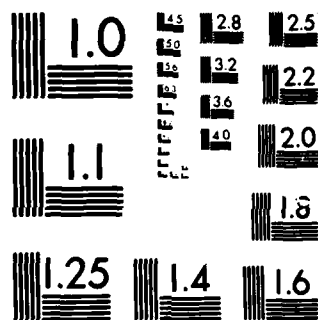
UNCLASSIFIED

DEC 85 AFIT/GCS/NA/85D-2

F/B 9/2

NL

										END			
										PRINTED			
										UTL			



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

```

12 For example: student.gpa would be how you reference the "gpa" float
12 variable within the "student" structure that I declared using the
12 first method of structure declaration. Whereas, student_1.gpa is
12 how to reference the "gpa" float variable within the "student_1" struc-
12 ture (of type "stu_rec") that was declared using the second method.

```

```

12 Let's look at another example to be sure you understand this concept.

```

```

13B:140

```

```

11Frame 140 T

```

```

12 * Structure Variable Access Continued *

```

```

12 struct employees {      ; This declaration sets up a "template" for a
12     int num_male;        ; structure of type "employees" as well as
12     int num_female;     ; declares "dep_1" to be a variable of that
12     int num_over_40;    ; type. This is a legal declaration that com-
12     int num_under_40;   ; bines both methods of structure declaration.
12 } dep_1;                ; I show it here to make you aware of its use.

```

```

12 The way in which you would reference the individual members of the
12 declared structure "dep_1" is as follows:

```

```

12 dep_1.num_male          ; Each of these individual variable members of
12 dep_1.num_female        ; the structure can be used as you would any
12 dep_1.num_over_40       ; variable of their individual type ("int").
12 dep_1.num_under_40      ; Let's now take a quick look at how you can
12                          ; initialize a structure.

```

```

13B:145

```

```

11Frame 145 T

```

```

12 *** Structure Initialization ***

```

```

12 A structure may be initialized by listing the member values after the
12 structure name declaration. The following two examples show how this
12 is done.

```

```

12 struct {                ; struct planes {
12     int tot_num;         ;     int tot_num;
12     int tot_maint;       ;     int tot_maint;
12     int tot_avail;       ;     int tot_avail;
12 } planes = {50,5,45};    ; };
12                          ; struct planes F_16 = {50,5,45};

```

```

13B:150

```

```

11Frame 150 QM

```

```

12Given the structure declaration: struct houses {
12     int num_white;
12     int num_green;
12     int num_brick;
12 } quarters = {165,139,127};

```

```

12Which of the following is a way to increase the "num_brick" variable to 137 ?

```

```

13A houses.num_brick += 10;

```

```

13B quarters.houses.num_brick += 10;

```

```

13C+ quarters.num_brick += 10;

```

```

13D houses.quarters.num_brick += 10;
14 Very good.
14 B:155
15ABD No. Answer "C" is the correct one.
15 B:155
15E "E" was not a given choice. Please try again.
15 B:150
11Frame 155 T
12 *** Topic Review ***
12
12 In this topic we have looked at the concept of a structure and we
12 examined two methods of declaring them.
12
12 We have seen how to access the individual members of a declared struc-
12 ture, and we also saw how you can initialize a structure when it is
12 declared. We have seen examples of what these structures look like
12 and how they can be used.
12
12 In the next topic area I will describe "structures within structures"
12 and "arrays of structures".
12
12 See you there!
12
12
12 *** This concludes this topic area. ***
13END
21Frame 300 T Structures and Arrays
22 *** Introduction ***
22
22 In this topic area I will describe how structures are used within
22 structures and how to declare and use an array of structures.
22
22 The uses for these two capabilities is unlimited to say the least.
22
22 The description of how to use these two capabilities is very straight-
22 forward, so this won't take long.
22
22 Note: Variable names in all CAPS are assumed to be declared constants.
22
22 Let's get to it.
23B:305
21Frame 305 T
22 *** Structures Within Structures ***
22
22 As you may have deduced by now, there is no restriction on the types of
22 variables used within a structure. Therefore, we can have a structure
22 that contains a variable that is itself a structure.
22
22 For example:
22
22 Declare "employee" * Declare "home" * Declare "wage_earner"

```

```

22 *****
22 struct employee {      * struct home {      * struct {
22     char f_name[FSIZE]; *   char street[SSIZE]; *   struct employee name;
22     char m_init;         *   char city[CSIZE]; *   struct home address;
22     char l_name[LSIZE]; *   long zip;           *   float wage;
22 };                      * };                   * } wage_earner;

```

23B:310

21Frame 310 T

22 * Structures Within Structures Continued *

22

22 We can now use the structure member operator (.) to gain access to a
22 specific member of our declared structure "wage_earner".

22

22 Given:

22

22 Structure "employee" * Structure "home" * Structure "wage_earner"

22 *****

```

22 struct employee {      * struct home {      * struct {
22     char f_name[FSIZE]; *   char street[SSIZE]; *   struct employee name;
22     char m_init;         *   char city[CSIZE]; *   struct home address;
22     char l_name[LSIZE]; *   long zip;           *   float wage;
22 };                      * };                   * } wage_earner;

```

22

22 wage_earner.name.m_init will access the character variable used for a
22 wage earners middle initial.

23B:315

21Frame 315 QM

22 Structure "employee" * Structure "home" * Structure "wage_earner"

22 *****

```

22 struct employee {      * struct home {      * struct {
22     char f_name[FSIZE]; *   char street[SSIZE]; *   struct employee name;
22     char m_init;         *   char city[CSIZE]; *   struct home address;
22     char l_name[LSIZE]; *   long zip;           *   float wage;
22 };                      * };                   * } wage_earner;

```

22

22 Given the above, which of the following is "not" a valid variable access
22 expression?

23A wage_earner.wage

23B wage_earner.address.zip

23C+ wage_earner.home.street

23D wage_earner.name.m_init

24 Right.

24 B:320

25ABD Wrong. Answer "C" is the correct response.

25 B:320

25E "E" was not a given choice, please try again.

25 B:315

21Frame 320 T

22 *** Arrays of Structures ***

22

22 Now that we have seen how to have structures within structures, let's
22 take a look at how to declare an array of structures.

```

22
22 First we need to declare a structure:
22
22 struct address {
22     char street[S_SIZE];
22     char city[C_SIZE];
22     long zip;
22 };
22
22 We can now declare an array of this type of structure:
22
22 struct address student[100];
22B:325
21Frame 325 T
22 * Arrays of Structures Continued *
22
22 The statement: "struct address student[100];" will allocate memory
22 space for 100 structures of type "address". Each of these structures
22 can now be accessed by using an array index and the structure member
22 operator (.).
22
22 For example:
22
22 student[49].zip will access the variable "zip" of the 50th structure
22 (of type "address") in the array "student".
22
22 student[9].city = "New York"; will assign the character string "New
22 York" to the character array "city" of
22 the 10th structure (of type "address")
22 in the array "student".
22B:330
21Frame 330 QP
22 Given the declaration: struct name {
22     char f_name[F_SIZE];
22     char m_init;
22     char l_name[L_size];
22 } roster[50];
22
22A "template" structure of type "name" is declared and an array of 50 of these
22 structures called "roster" is declared as well. (True or False)
22Y
24 Very good.
24 B:335
25 Wrong. This is one way we have seen to combine the two methods of struc-
25 ture declaration.
25 B:335
21Frame 335 T
22 *** Topic Review ***
22
22 In this topic we have looked at how structures are declared and used
22 within other structures and we saw how to declare and use an array
22 of structures.

```

```

22
22   Although we didn't look at very many or very involved examples of the
22   uses of these two capabilities, I think that it is enough to introduce
22   you to their use and will spark your ingenuity for programming appli-
22   cations. The rest of this lesson will discuss some other ways of work-
22   ing with structures in C programming.
22
22   In the next topic area (3) I will describe and show examples of how
22   to use pointers to structures. In topic area four I will discuss
22   how to pass structure data between functions.
22
22           *** This concludes this topic area. ***
23END
31Frame 500 T   Structures and Pointers
32   *** Introduction ***
32
32   In lesson four we saw that a "pointer" is actually a variable that con-
32   tains the address of where some other variable resides in memory.
32
32   In this topic area I will describe how pointers are used to access
32   structures and their members. We will take a look at a couple examples
32   to help see this fairly straightforward technique.
32
32   Let's get started!
33B:505
31Frame 505 T
32   *** Pointers to Structures ***
32
32   We have seen that given a structure declaration such as:
32
32   struct income {
32       float gross;
32       float fitw;
32       float s_tax;
32       float fica;
32   } pay;
32
32   This declares a "template" structure of type "income" and also declares
32   a variable "pay" to be of that type.
32
32   As we have seen, we can now access the individual members of the vari-
32   able "pay" by using the "structure member operator" (.). For example:
32   pay.gross will access the variable "gross" within the structure "pay".
33B:510
31Frame 510 T
32   * Pointers to Structures Continued *
32
32   Let's now look at how we can use pointers to access the structure and
32   its members.
32
32   Given the structure declaration: struct income {

```

```

32         float gross;
32         float fitw;
32         float s_tax;
32         float fica;
32     } pay;
32

```

32 We can use the pointer declaration: struct income *p_pay; to declare a pointer "p_pay" that points to a structure of type "income".

32 Using the statement: p_pay = &pay; we assign the starting address of variable "pay" (of structure type "income") to variable "p_pay".

33B:515

31Frame 515 OM

```

32Given the structure declaration: struct address {
32                                char street[S_SIZE];
32                                char city[C_SIZE];
32                                long zip;
32                                } home;
32

```

32Which of the following will assign the starting address of the structure "home" to the pointer "p_home" ?

- 33A p_home = home;
- 33B p_home = address;
- 33C+ p_home = &home;
- 33D p_home = &address;

34 Right.

34 B:520

35ABD Wrong. Response "C" is the correct one.

35 B:520

35E "E" was not a given choice. Please try again.

35 B:515

31Frame 520 T

32 * Pointers to Structures Continued *

32 Now that we have defined a pointer to the structure "pay", we need to learn how to use this pointer to access the members of the structure.

32 The way in which this is done in C is through the use of the a special operator which is composed of a minus and greater than sign "->".

32 For example, if we wish to access the variable "gross" of the structure "pay" in our example, we could use the expression: p_pay->gross

32 This, of course, would be used in a statement such as:

32 p_pay->gross = gross_pay; which would store the value of "gross_pay" in the memory location represented by "gross" within structure "pay".

33B:525

31Frame 525 T

32 * Pointers to Structures Continued *

32

32 The special operator -> is provided as a shorthand way of accomplish-
32 ing the same thing that the unary operator * does.

32 The statement we just saw, p_pay->gross = gross_pay; , could have been
32 just as easily written as: (*p_pay).gross = gross_pay; and would
32 have the same result.

32 The problem with using the unary operator * (asterisk) is that it
32 has a lower precedence than the structure member operator . (period).
32 Hence, you must use parentheses to ensure proper execution.

32 With this in mind it is easy to see that using the provided special
32 operator -> is easier and clearer.

33B:530

31Frame 530 QP

32Given the structure declaration: struct address {
32 char street[S_SIZE];
32 char city[C_SIZE];
32 long zip;
32 } home;

32The variable "zip" can be accessed by using the expression: p_home->zip
32Provided "p_home" has been declared a pointer to type "address".

32(True or False)

33N

34 Very good. The correct expression is: p_home->zip.

34 B:535

35 No. The correct expression is: p_home->zip.

35 B:535

31Frame 535 T

32 * Pointers to Structures Continued *

32 As a quick review.

32 If you have a structure 32 declaration like:	32 And a pointer variable 32 declaration like:	32 You can access 32 the individual 32 structure members 32 with expressions:
--	---	--

32 struct income {	32 struct income *p_pay;	32 p_pay->gross
32 float gross;	32 p_pay = &pay;	32 p_pay->fitw
32 float fitw;		32 p_pay->s_tax
32 float s_tax;		32 p_pay->fica
32 float fica;		
32 } pay;		

33B:540

31Frame 540 T

32 *** Topic Review ***

32 In this topic area we have looked at how pointers to structures are
32 declared and how to access the individual members of a structure
32 using a declared pointer.

```

32
32 We have seen a couple examples to help illustrate this technique.
32
32 In the next topic area I will describe how to pass structure data
32 between functions.
32
32 Hope to see you there!
32
32
32 *** This concludes this topic area. ***
32END
41Frame 700 T Structures and Functions
42 *** Introduction ***
42
42 In this topic area I will describe how structure data is passed between
42 functions.
42
42 We have seen already how to pass variables as well as pointers between
42 functions. Passing structure data is done in much the same way. We
42 will look at a few examples to help illustrate this concept.
42
42
42 Let's get started!
43B:705
41Frame 705 T
42 *** Passing Structure Data ***
42
42 Using the structure we defined in the last topic area:
42
42 struct income {
42     float gross;
42     float fitw;
42     float s_tax;
42     float fica;
42 } pay;
42
42 One way to pass the data contained in the structure to a called func-
42 tion is to pass the structure members individually. For example:
42
42 compute(pay.gross,pay.fitw,pay.s_tax,pay.fica);
42
42 Calls function "compute" and passes the four members of structure "pay".
43B:710
41Frame 710 T
42 * Passing Structure Data Continued *
42
42 The called function would look something like the following in order
42 to receive and use the passed variables:
42
42 float compute(gross,fitw,s_tax,fica)
42     float gross,fitw,s_tax,fica;

```

```

42  {
42      take_home_pay = gross - (fitw + s_tax + fica);
42      return(take_home_pay);
42  }
43B:715
41Frame 715 T
42  * Passing Structure Data Continued *
42
42  A second way to pass the structure data to the called function is to
42  pass the entire structure. For example:
42
42  compute(pay); will pass the address of the beginning of structure
42                  "pay" to function "compute".
42
42  The called function would look something like the following in order
42  to receive and use the passed structure address.
42
42  float compute(p_data)
42      struct income p_data;
42  {
42      t_h_p = p_data.gross - (p_data.fitw + p_data.s_tax + p_data.fica);
42      return(t_h_p);
42  }
43B:720
41Frame 720 T
42  * Passing Structure Data Continued *
42
42  A third way to pass the structure data to the called function is to
42  pass a pointer to the structure. For example, if you have a structure
42  defined as:
42      struct income {
42          float gross;
42          float fitw;
42          float s_tax;
42          float fica;
42      } pay;
42
42  Define a pointer variable with the statement: struct income *p_pay;
42
42  Assign the address to the pointer variable: p_pay = &pay;
42
42  Then call the function: compute(p_pay);
43B:725
41Frame 725 T
42  * Passing Structure Data Continued *
42
42  The called function would look something like the following in order
42  to receive and use the passed pointer variable:
42
42  float compute(pntr)
42      struct income *pntr;

```

```

42  {
42      t_h_p = pntr->gross - (pntr->fitw + pntr->s_tax + pntr->fica);
42      return(t_h_p);
42  }
43B:730
41Frame 730 QM
42Which of the following is not one of the three ways of passing structure
42data to a called function?
43A Pass structure members individually.
43
43B+ Pass the structure template name.
43
43C Pass the entire structure.
43
43D Pass a pointer to the structure.
44 Your right.
44 B:735
45ACD Wrong. Answer "B" is not a valid way to pass structure data.
45 B:735
45E "E" was not a give choice. Please try again.
45 B:730
41Frame 735 T
42 *** Lesson Five Summary ***
42
42 Well, we have come to the end of lesson five. If you have seen the
42 four subject topics in this lesson, you should now be ready to take
42 the final test. If you feel that you don't understand something well
42 enough to pass the test, please retake the topic that is giving you
42 problems.
42
42 Topic 1 gave an introduction to structures and their use.
42
42 Topic 2 gave a description of structures within structures and arrays
42 of structures.
42
42 Topic 3 gave a description of how pointers to structures are used.
42
42 Topic 4 described how structure data is passed between functions.
42END
51Frame 900 TT TEST OVER LESSON 5
52 Welcome to the final test of lesson five. This test consists of seven
52 questions over material presented in the previous four topic areas.
52
52 In order to successfully complete this lesson, you must achieve a
52 minimum score of 71.4% (five out of seven questions correct).
52
52 If you miss a question, the correct answer will not be shown. It is
52 up to you to research the correct answer.
52
52 Well, enough said. Let's get on with it. Good luck!
53B:905

```

51Frame 905 QM

521. Which of the following can be used to declare a structure?

53A struct structure_tag { variable declarations };

53B struct { variable declarations }; structure name;

53C struct { variable declarations } structure name;

53D None of the above.

53E+ Both "A" and "C" above.

54 Right. (1,110 & 125)

54 B:910

55ABCD Wrong. (1,110 & 125)

55 B:910

51Frame 910 QM

522. Given the structure declaration: struct houses {

52 int num_wood;

52 int num_brick;

52 int num_stucco;

52 } resident;

52

52Which of the following is a way to access the variable "num_brick" ?

53A houses.num_brick

53B houses.resident.num_brick

53C+ resident.num_brick

53D resident.houses.num_brick

54 Right. (1,135-140)

54 B:915

55ABD Wrong. (1,135-140)

55 B:915

55E "E" was not one of your choices.

55 B:910

51Frame 915 QF

523. In the C programming language there is no provision for the use of

52structures within structures because it would require too much memory

52overhead. (True or False)

53N

54 Right. (2,305)

54 B:920

55 Wrong. (2,305)

55 B:920

51Frame 920 QM

524. Given the structure declaration: struct address {

52 char street[S_SIZE];

52 char city[C_SIZE];

52 long zip;

52 };

52

52Which of the following is a way to declare an array of 50 such structures?

53A array_of_address struct address[50];

53B+ struct address array_of_address[50];

53C struct array_of_address address[50];

53D address[50] struct array_of_address;

54 Right. (2,320-325)

54 B:925

55ACD Wrong. (2,320-325)

55 B:925

51Frame 925 QP

```
525. Given the structure declaration: struct address {  
52      char street[S_SIZE];  
52      char city[C_SIZE];  
52      long zip;  
52      } home;  
52
```

52And the pointer declaration: struct address *p_home;

52

52The statement: p_home = &home; will assign the starting address of the
52structure "home" (of type "address") to the pointer "p_home".

52(True of False)

53Y

54 Right. (3,510)

54 B:930

55 Wrong. (3,510)

55 B:930

51Frame 930 QM

```
526. Given the declaration: struct name {  
52      char f_name[F_SIZE];  
52      char m_init;  
52      char l_name[L_size];  
52      } roster[50];  
52
```

52Which of the following expressions can be used to access the variable
52"m_init" (Assume pointer "p_roster" has been properly declared.)?

52A+ p_roster->m_init

52B p_roster->m_init

52C p_roster->roster.m_init

52D p_roster->name->roster.m_init

54 Right. (3,515)

54 B:935

55BCD Wrong. (3,515)

55 B:935

55E "E" was not one of your choices.

55 B:930

51Frame 935 QM

527. Which of the following is not one of the three ways of passing structure
52data to a called function?

52A Pass structure members individually.

52B Pass the entire structure.

52C+ Pass the structure template name.

52D Pass a pointer to the structure.

54 Right. (4,705,715,720)

54 B:940

55ABD Wrong. (4,705,715,720)

55 B:940

55E "E" was not one of your choices.

55 B:935

51Frame 940 T

52 *** End of Lesson Material ***

52

52 This marks the end of lesson number five. I hope that it was of some
52 benefit to you. I am looking forward to seeing you in lesson number
52 six. I hope that you didn't have too much trouble with the material
52 presented in this lesson. If you did, please voice your comments to
52 your training monitor who will in turn contact the CAI Plans Branch
52 at Keesler AFB, MS.

52

52 Well, let's take a look at how you did with the test ...

53END

File "LESSON6"

```
#      WW      WW  EEEEEEEE  LL      CCCCCC  000000  MMM  MMM  EEEEEEEE
#      WW WW WW  EE      LL      CC      00      00  MMM  MMM  EE
#      WW WW WW  EEEEE  LL      CC      00      00  MM MM MM  EEEEE
#      WWW  WWW  EE      LL      CC      00      00  MM MM MM  EE
#      WWW  WWW  EEEEEEEE  LLLLLLLL  CCCCCC  000000  MM      MM  EEEEEEEE
#
#
#
#      TTTTTTTTTT  00000000
#      TT      00      00
#      TT      00      00
#      TT      00      00
#      TT      00000000
#
#
#      LL      EEEEEEEE  SSSSSS  SSSSSS  000000  NN      NN      6666
#      LL      EEEEEEEE  SSSSSSSS  SSSSSSSS  00000000  NNN      NN      6666
#      LL      EE      SSS      SSS      00      00  NNNN  NN      666
#      LL      EEEEE  SSSS      SSSS      00      00  NN NN NN      666
#      LL      EEEEE  SSSS      SSSS      00      00  NN NN NN      66666666
#      LL      EE      SSS      SSS      00      00  NN  NNNN      666  66
#      LLLLLLLL  EEEEEEEE  SSSSSSSS  SSSSSSSS  00000000  NN      NN      666 666
#      LLLLLLLL  EEEEEEEE  SSSSSS  SSSSSS  000000  NN      NN      66666
```

THE LESSON YOU ARE ABOUT TO TAKE CONTAINS INTRODUCTORY INFORMATION ON
INPUT AND OUTPUT CAPABILITIES OF THE C PROGRAMMING LANGUAGE.

THE LESSON CURRENTLY CONSISTS OF FIVE TOPICS.

The Lesson Breakdown Is As Follows:

Topic 1: Getchar and Putchar - This topic gives a description of the use
of the standard I/O functions "getchar" and "putchar".
(Approx. time = 5 min.)

Topic 2: Getline - This topic gives a description of the use of the stan-
dard input function "getline" and presents an example "getline"
function. (Approx. time = 5 min.)

Topic 3: Scanf - This topic gives a description and examples of the stan-
dard input function "scanf". (Approx. time = 15 min.)

Lesson Breakdown Continued:

Topic 4: Printf - This topic gives a description and examples of the stan-
dard output function "printf". (Approx. time = 10 min.)


```
# Topic 5: Lesson 6 Test - This is the lesson test over items that have
# been presented in the previous four lesson topics.
# (Approx. time = 5 min.)
#
```

```
# TOTAL LESSON TIME IS APPROXIMATELY 40 MINUTES.
#
```

```
# I hope that you enjoy it!
!
```

```
*****
```

```
*
*          SELECT THE TOPIC YOU WISH TO TAKE FROM THE FOLLOWING:
*
```

```
*****
```

STATUS	TOPIC #	TOPIC TITLE
	1	Getchar and Putchar
	2	Getline
	3	Scanf
	4	Printf
	5	Test Over Lesson 6

```
*****
* NOTE: A "STATUS" OF "+" INDICATES TOPIC SUCCESSFULLY COMPLETED.
*
*****
```

```
11Frame 100 T Getchar and Putchar
```

```
12 *** Introduction ***
```

```
12
12 Input/Output (I/O) is "not" a part of the C programming language.
12 Statements such as Print, Write, or Read are "not" available for use.
12
12 The way in which you compensate for C's lack of I/O capability is to
12 make use of library functions supplied by the C compiler's manufacturer.
12
12 The types of functions that are provided with a specific C compiler
12 vary from manufacturer to manufacturer, so it is suggested that you
12 review your C compiler's documentation in order to determine what
12 functions you can make use of.
12
12 In this topic area we will take a look at some basic I/O functions
12 that most manufacturers provide.
```

```
13B:105
```

```
11Frame 105 T
```

```
12 * Introduction Continued *
```

```
12
```

12 In order for you to have access to the standard I/O functions provided with your C compiler you may need to include a header file that contains the definitions and declarations needed by the I/O functions.

12 The file name that you include depends on the compiler you are using. Typically the include statement will look something like the following:

12 #include <stdio.h> OR #include <bdscio.h>

12 Please check your compiler's documentation for the proper header file to be included, if any.

12 In this lesson topic we will be discussing how to use the standard I/O functions "getchar" and "putchar". We will see examples of how these two functions are called and what they do. Let's get started.

13B:110

11Frame 110 T

12 *** Getchar ***

12 The function "getchar" is used to read one character at a time from the standard input device. The standard input device is by default the users terminal keyboard.

12 Note: The standard input device can be changed on most systems, but how this is done will not be discussed in this course.

12 The format of the call to the function "getchar" is as follows:

12 c = getchar(); Where "c" is any variable of type "int".

12 What was that? Variable "c" is of type "int"! Well, that just doesn't sound right. Let's look at this a little closer.

13B:115

11Frame 115 T

12 * Getchar Continued *

12 The requirement that the variable that receives the character returned by the function "getchar" be of type "int" stems from the fact that "getchar" is a function that returns an integer value.

12 The only time you would run into problems in making the variable "c" a "char" type is if you were trying to detect an end of file condition. The reason for this is that EOF is typically equal to -1, which is of course an integer.

12 Thus, when the EOF is encountered it must be read into a variable of type "int".

13B:120

11Frame 120 T

12 * Getchar Continued *

12 For example, the following program will "not" work.

```

12
12     main() {
12         char c;
12         while ((c = getchar()) != EOF)
12             < some statement to deal with variable "c" >;
12     }

```

12 The proper way to write the program is:

```

12     main() {
12         int c;
12         while ((c = getchar()) != EOF)
12             < some statement to deal with variable "c" >;
12     }

```

13B:125

11Frame 125 T

12 * Getchar Continued *

12 As another example, the following program will work since no check is
12 made against "EOF".

```

12     main() {
12         char c;
12         while ((c = getchar()) != '\n')
12             < some statement to deal with variable "c" >;
12     }

```

12 Here the terminating condition is when "c" is equal to the "newline"
12 C escape sequence. As you can see, the requirement for the receiving
12 variable of the function "getchar" to be of type "int" is not without
12 exception. Just be aware of the fact that "getchar" returns an "int"
12 type and this may cause you a problem if the receiving variable is not
12 of the same type.

13B:130

11Frame 130 QP

12The "getchar" function is used to read one character at a time from standard
12input to the executing C program. (True or False)

13Y

14 Right.

14 B:135

15 Wrong. Wake up!

15 B:135

11Frame 135 T

12 *** Putchar ***

12 The function "putchar" is used to write one character at a time to
12 the standard output device. The standard output device is by default
12 the users terminal screen.

12 Note: The standard output device can be changed on most systems, but
12 how this is done will not be discussed in this course.

12

12 The formats of the call to the function "putchar" is as follows:

12

12 putchar(c); Where c is any character variable.

12

12 putchar('c'); Where 'c' is any character constant.

12

12 putchar("\c"); Where \c is any C escape sequence.

13B:140

11Frame 140 T

12 * Putchar Continued *

12

12 For example:

12

```
12 main() {  
12     putchar('I');  
12     putchar(' ');  
12     putchar('l');  
12     putchar('i');  
12     putchar('k');  
12     putchar('e');  
12     putchar(' ');  
12     putchar('C');  
12     putchar('.'); }
```

12

12 This program will write the sentence: I like C. to the standard

12 output device (terminal screen).

13B:145

11Frame 145 T

12 * Putchar Continued *

12

12 As another example:

12

```
12 main() {  
12     char string[] = "I like C.";  
12     for (i = 0; string[i] != '\0'; i++)  
12         putchar(string[i]);  
12 }
```

12

12 This program will also write the sentence: I like C. to the standard

12 output device (terminal screen). The loop terminating expression will

12 become "true" when the end-of-string marker (\0) is encountered.

13B:150

11Frame 150 QM

12 Which of the following is "not" a correct way to use the "putchar" function?

13A putchar(c); Where c is any character variable.

13B+ putchar(*c); Where *c is a pointer to any character array.

13C putchar('c'); Where 'c' is any character constant.

13D putchar("\c"); Where \c is any C escape sequence.

14 Very good.

14 B:155

15ACD No. Answer "B" is the correct one.

15 B:155

15E "E" was not a given choice. Please try again.

15 B:150

11Frame 155 T

12 *** Combination Example ***

12

12 This example shows how you can combine both the "getchar" and "putchar"
12 functions to read & write a line of text from/to the standard I/O
12 device.

12

```
12     main() {  
12         char c;  
12         while ((c = getchar()) != '\n')  
12             putchar(c);  
12     }
```

12

12 This program will terminate when the user hits the "Return" key at the
12 end of his/her typed line.

13B:160

11Frame 160 T

12 *** Topic Review ***

12

12 In this topic we have looked at the standard I/O functions "getchar"
12 and "putchar".

12

12

12 We have seen a few examples of how to access and use these functions
12 and discussed a couple of things to be aware of in their use.

12

12

12 In the next topic area I will describe the I/O function "getline" and
12 give a few examples of its use.

12

12 See you there!

12

12

12

12

12

12

12

12

12

12

12

12

12

12

12

12

12

12

12

12

12

12

12

12

12

12

12

12

12

12

*** This concludes this topic area. ***

13END

21Frame 300 T Getline

22 *** Introduction ***

22

22

22

22

22

22

22

22

22

22

22

22

22

22

22

22

22

22

22

22

22

22

22

22

In this topic area I will describe the I/O function "getline".

This function is used to read in one line of input from the standard
input device (users terminal keyboard). In addition to reading a
line of input, the "getline" function also keeps track of how many
characters were read in.

We saw in the last topic area how to accomplish the reading of a line
of input using the "getchar" function, but as you can well imagine, if
you need to do this task in several points in your program it would pay
to have a separate function defined which you could call.

22 Most C compilers have this function as part of its I/O library, but
 22 just in case your compiler manufacturer didn't include it, I will
 22 present a version of "getline" that you can use in your programs.

23B:305
 21Frame 305 T
 22 *** Getline ***
 22
 22 The format of the call to the function "getline" is as follows:
 22
 22 n = getline(input_line,80);
 22
 22 Where "n" is any variable of type "int", "input_line" is a character
 22 array, and "80" is the maximum length of the array. When the above
 22 statement is executed the "getline" function will read a line of input
 22 from the users terminal keyboard. The above call will read in at most
 22 78 characters. If the user were to type 78 characters and then hit the
 22 "Return" key, the actual contents of the "input_line" array would be as
 22 follows:
 22
 22 input_line[0] thru input_line[77] = characters (78 characters)
 22 input_line[78] = \n (end of line character)
 22 input_line[79] = \0 (end of string marker)

23B:310
 21Frame 310 T
 22 * Getline Continued *

22 As I stated before, the "getline" function will keep track of the
 22 number of characters it reads in. What I didn't mention is that
 22 it will return this number to the calling function if so desired.
 22
 22 In our example statement: n = getline(input_line,80);
 22
 22 The variable "n" (of type "int") is where the number of characters
 22 read in is stored. This number will include the 78 characters of
 22 user input and the end of line character, but not the end of string
 22 marker. For our example this would give us a total count of 79.
 22
 22 How you use this number, if at all, depends on your programs applica-
 22 tion.

23B:315
 21Frame 315 QM
 22 Given the function call statement: n = getline(input_line,80);
 22
 22 Which of the following is "not" true.
 23A "n" must be a variable of type "int".
 23B+ "getline" will return two values "n" and "input_line".
 23C "input_line" must be a character array.
 23D "80" is the maximum input line size.
 24 Right. "getline" will return an integer value to "n", but the array
 24 "input_line" is passed as a pointer to array position input_line[0].
 24 B:320
 25ACD Wrong. Answer "B" is the correct response. "getline" will return an

25 integer value to "n", but the array "input_line" is passed as a pointer to
25 array position input_line[0].

25 B:320

25E "E" was not a given choice, please try again.

25 B:315

21Frame 320 T

22 * Getline Continued *

22

22 Let's take a look at a sample program that uses the function "getline".

22

```
22 main() {  
22     char input_line[80];  
22     getline(input_line,80);  
22     i = 0;  
22     while (input_line[i] != '\0') {  
22         putchar(input_line[i]);  
22         i++;  
22     }  
22 }
```

22

22 This program will read in one line of input from the users terminal
22 keyboard and print the stored line (one character at a time) on the
22 users terminal screen.

23B:325

21Frame 325 T

22 * Getline Continued *

22

22 Now that we have seen how to use the "getline" function that is usually
22 provided with your C compiler by the manufacturer, let's take a look at
22 how you can define your own version of the "getline" function.

22

22

22 The following will perform the same as the "getline" function we have
22 just looked at and can be included in your programs if the "getline"
22 function is not available.

23B:330

21Frame 330 T

22 * Getline Continued *

22

```
22 getline(in_ln,max)  
22 char in_ln[];  
22 int max;  
22 {  
22     int i,c;  
22     for (i = 0; i < (max-1) && (c = getchar()) != EOF && c != '\n'; i++)  
22         in_ln[i] = c;  
22     if (c == '\n')  
22         in_ln[i++] = c;  
22     in_ln[i] = '\0';  
22     return(i);  
22 }
```

22

23B:335

21Frame 335 QP

22Given the function call statement: getline(input_line,35);

22

22The maximum number of characters that will be read by the function "getline"
22is 35. (True or False)

23N

24 Very good. 34 characters can be read. One character is used to store the
24 end of string marker.

24 B:340

25 Wrong. 34 characters can be read. One character is used to store the
25 end of string marker.

25 B:340

21Frame 340 T

22 *** Topic Review ***

22

22 In this topic we have looked at the I/O function "getline" which may
22 or may not be included with your C compiler's standard I/O library.

22

22 We have seen a few examples of how to access and use this function
22 and we saw a version of the function that you can include in your
22 program if it is not available with your compiler.

22

22

22 In the next topic area I will describe the I/O function "scanf" and
22 give a few examples of its use.

22

22 See you there!

22

22

22

*** This concludes this topic area. ***

23END

31Frame 500 T Scanf

32 *** Introduction ***

32

32 In this topic area I will describe the I/O function "scanf".

32

32 This function is used to read characters from the standard input
32 device (users terminal keyboard) and do some sort of conversion on
32 the read characters. In essence the function is used to do format-
32 ted input.

32

32 We saw in the last topic area how to accomplish the reading of a line
32 of input using the "getline" function, but if the input you wish to
32 read is not composed of just characters you would be hard put to store
32 the input in their intended form.

32

32 All C compilers should have the function "scanf" as part of its I/O
32 library. Please check your compiler's documentation to be sure of
32 this functions availability.

33B:505

31Frame 505 T

32 *** Scanf ***

32 The format of the "scanf" function call is composed of two parts:
 32 a format control string and the pointer arguments.
 32
 32 A skeleton of the function call looks like this:
 32
 32 scanf("format control string", &arg_1, &arg_2, ..., &arg_n);
 32
 32 The format control string will be described in detail shortly. The
 32 arguments following the string must be pointers to the memory loca-
 32 tions where the read in arguments are to be stored.
 32
 32 It is a fairly common mistake to try and read values into a variable
 32 by just specifying the variable name. This can not be done since
 32 "scanf" is a function and as such can only return one value. Thus,
 32 you must somehow pass it the address of where the variable is stored.

33B:510

31Frame 510 T

32 * Scanf Continued *

32 The format control string will usually contain the conversion speci-
 32 fications to be applied to the input sequences read from the input
 32 device.

32 The format control string begins with a percent sign (%) and ends
 32 with either a conversion character or character class.

32 The following is a verbal description of what is allowed for use in
 32 the format control string:

32 A "percent sign" followed by an "argument suppression character"
 32 followed by an "integer field width specifier" followed by a "length
 32 modification character" followed by a "conversion character or char-
 32 acter class".

33B:515

31Frame 515 QP

32The format of the "scanf" function call is composed of two parts:
 32a "format control string" and the "pointer arguments". (True or false)
 33Y

34 Right. I'm glad your paying attention.

34 B:520

35 Wrong. That is a true statement.

35 B:520

31Frame 520 T

32 * Scanf Continued *

32 Let's now look at each part of the "format control string" of the
 32 scanf function call.

32 The "format control string" is made up of individual conversion
 32 specifications. Each of these conversion specifications "must"
 32 begin with a "percent sign" (%).

32
 32 The next (optional) character is an "argument suppression character".
 32 This character is an asterisk (*) and indicates that the next input
 32 field is to be skipped. Thus, no assignment is made into the corre-
 32 sponding input argument.
 32
 32 The next (optional) part of the "string" is an "integer field width
 32 specifier" which is used to specify the maximum field width of the
 32 input.
 33B:525
 31Frame 525 T
 32 * Scanf Continued *
 32
 32 The next (optional) part of the "string" is the "length modification
 32 character". This character can be one of two letters: l or h .
 32 These two letters can only be used in conjunction with certain "con-
 32 version characters" as will be described forthwith.
 32
 32 The last part of the "string" is the "conversion character or char-
 32 acter class". The "conversion character" can be one of 13 different
 32 characters.
 32
 32 I will now give a brief description of each of these characters.
 33B:530
 31Frame 530 T
 32 * Scanf Continued *
 32
 32 d = decimal integer (argument should point to "int" variable type.)
 32 o = octal integer (argument should point to "int" variable type.)
 32 x = hexadecimal integer (argument should point to "int" variable type.)
 32
 32 D = decimal integer (argument should point to "long" variable type.)
 32 O = octal integer (argument should point to "long" variable type.)
 32 X = hexadecimal integer (argument should point to "long" variable type.)
 32
 32 e or f = floating point number (argument should point to "float" vari-
 32 able type.)
 32
 32 E or F = floating point number (argument should point to "double" vari-
 32 able type.)
 33B:535
 31Frame 535 T
 32 * Scanf Continued *
 32
 32 c = character (argument should point to "character" variable type.)
 32
 32 s = string (argument should point to "character array" variable type.)
 32
 32 % = percent sign is expected as the next input character.
 32
 32 As a refresher: Integer input: d, o, x, D, O, or X
 32

```

32          Floating point input:  e, f, E, or F
32
32          Character input:  c
32
32          String input:  s
32
32          Percent sign input:  %
33B:540
31Frame 540 T
32      * Scanf Continued *
32
32      Let's look at a couple of examples involving the "scanf" function call.
32
32          scanf("%d%f",&int_var,&float_var);
32
32      The above call will read from standard input (users terminal keyboard)
32      two numbers of the types "integer" and "floating point real".
32
32      The users typed input numbers would be of the form:  23 45.78
32
32      The "scanf" function will read into the first aurgment ("int_var")
32      until a "white space" character or a character that is incompatiable
32      with the specified "format control string" is encountered.
32
32      Note:  A "white space" character is defined as a "blank", "tab" (\t),
32      or "newline" (\n).
33B:545
31Frame 545 T
32      * Scanf Continued *
32
32      As another example:  scanf("%s%kd%",s_array,&int_var);
32
32      This "scanf" call will read a "string", "integer", and "percent sign".
32
32      The users input would look something like this:  Tax = 5%
32
32      The function "scanf" will read the word "Tax" into the array "s_array",
32      then skip the character "=", then read the integer "5", and finally
32      read the "percent sign".  No space is needed after the "5" in the users
32      input since the "percent sign" is not compatiabile with the "%d" format
32      control string.  The "percent sign" is not stored anywhere.
33B:550
31Frame 550 QM
32Given the function call:  scanf("%d%f%s%c",&w,&x,y,&z);
32
32Which of the following variables will contain a number with a decimal point?
33A  w
33B+ x
33C  y
33D  z
34 Right.
34 B:555

```

35ACD Wrong. Response "B" is the correct answer.

35 B:555

35E "E" was not one of your choices, please try again.

35 B:550

31Frame 555 T

32 * Scnf Continued *

32

32 One more point on the "format control string" that I promised to
32 talk about, namely the "length modification character".

32

32 As I mentioned, this optional character can be either the letter
32 l or the letter h. The "length modification character" can only
32 be used with certain "conversion characters".

32

32 You may use the letter l with the conversion characters d, o, or x
32 to indicate that the value being read in is to be stored in a "long"
32 rather than "int" variable type. i.e., scanf("%ld",&l_int);

32

32 You may use the letter h with the conversion characters d, o, or x
32 to indicate that the value being read in is to be stored in a "short"
32 rather than "int" variable type. i.e., scanf("%hd",&s_int)

33B:560

31Frame 560 T

32 * Scnf Continued *

32

32 As I mentioned during the description of the format of the "scanf"
32 function call, the "format control string" begins with a percent sign
32 and ends with either a "conversion character" or "character class".

32

32 We have seen what the "conversion character" is, but we still need
32 to cover the "character class".

32

32 A "character class" is identified by a set of brackets [] following
32 the percent sign. The "character class" is used in conjunction with
32 a character array argument.

32

32 Let's look at two examples to demonstrate the use of "character class".

33B:565

31Frame 565 T

32 * Scnf Continued *

32

32 Example #1: scanf("%[abcdefghijklmnopqrstuvwxyz]",valid_letters);

32

32 In this example an input string is read until a letter is encountered
32 that "is not" in the "character class" specified. The character array
32 "valid_letters" must be big enough to hold the read in input string.

32

32 Example #2: scanf("%[^abcdefghijklmnopqrstuvwxyz]",valid_letters);

32

32 An alternate form of the "character class" uses a circumflex (^).
32 When this form is used, the valid input becomes any character not
32 specified in the "character class". Therefore, for example #2 above,

32 the input string will be read until a letter is encountered that "is"
32 in the "character class" specified.

33B:570

31Frame 570 T

32 *** Topic Review ***

32

32 In this topic we have looked at the I/O function "scanf" which is
32 usually included with your C compiler's standard I/O library.

32

32 We have seen a few examples of how to access and use this function
32 and discussed many of the special features of the function.

32

32

32 In the next topic area I will describe the I/O function "printf" and
32 give a few examples of its use.

32

32 See you there!

32

32

32

32 *** This concludes this topic area. ***

33END

41Frame 700 T Printf

42 *** Introduction ***

42

42 In this topic area I will describe the I/O function "printf".

42

42 This function is used to convert and print specified arguments to
42 the standard output device (users terminal screen). In essence the
42 function is used to do formatted output.

42

42 We saw in the last topic area how to accomplish formatted input by
42 using the "scanf" function. We will now cover how to accomplish the
42 task of producing output from your C program in any form you like.

42

42 All C compilers should have the function "printf" as part of its I/O
42 library. Please check your compiler's documentation to be sure of
42 this functions availability.

43B:705

41Frame 705 T

42 *** Printf ***

42

42 The format of the "printf" function call is composed of two parts:
42 a format control string and the arguments.

42

42 A skeleton of the function call looks like this:

42

42 printf("format control string", arg_1, arg_2, ..., arg_n);

42

42 The format control string will be described in detail shortly.

42 The arguments following the string have two important restrictions:

42

42 1. Their "type" must agree with the corresponding conversion

42 control character within the "format control string".
42
42 2. The number of arguments must agree with the number of con-
42 version control specifications in the "format control string".
43B:710
41Frame 710 T
42 * Printf Continued *
42
42 The format control string will usually contain the conversion speci-
42 cations to be applied to the output sequences being printed to the
42 output device.
42
42 However, you may also use the "printf" function to print character
42 sequences "character for character".
42
42 For example: printf("C is GREAT"); will print: C is GREAT
42
42 The format control string usually begins with a percent sign (%) and
42 ends with a conversion character, but can begin with C character es-
42 cape sequences.
42
42 For example: printf("\n\t%d",arg_1); will execute a "new line" and
42 a "tab", then print an integer.
43B:715
41Frame 715 QP
42The "printf" function call: printf("\nI Love C"); will execute a "new line"
42and then print the character sequence: I Love C (True or false)
43Y
44 Right. Good work!
44 B:720
45 Sorry, that is a true statement.
45 B:720
41Frame 720 T
42 * Printf Continued *
42
42 The following is a verbal description of what is allowed for use in
42 the "format control string" in addition to the "escape sequences".
42
42 A "percent sign" followed by a "minus sign" followed by an "integer
42 field width specifier" followed by a "period" followed by a "integer
42 precision specifier" followed by a "length modification character"
42 followed by a "conversion character".
42
42
42 Let's now look at each part of the "format control string" of the
42 "printf" function call.
43B:725
41Frame 725 T
42 * Printf Continued *
42
42 The "format control string" is made up of individual conversion
42 specifications. Each of these conversion specifications "must"

```

42 begin with a "percent sign" (%).
42
42 The next (optional) character is a "minus sign". The minus sign, if
42 present, indicates that the corresponding argument is to be printed
42 left justified in its field. If no minus sign is present then the
42 argument is printed right justified.
42
42 The next (optional) part of the "string" is an "integer field width
42 specifier" which is used to specify the minimum field width in which
42 the converted argument is to be printed.
43B:730
41Frame 730 T
42 * Printf Continued *
42
42 The next (optional) part of the "string" is a "period". The period
42 is used to separate the "integer field width specifier" from the next
42 field of the "format control string".
42
42 The next (optional) part of the "string" is an "integer precision
42 specifier". This is used to specify the maximum number of digits to
42 be printed to the right of the decimal point (in the case of "double
42 and float" argument types) or the maximum number of characters (in
42 the case of a "character string" argument).
42
42 The next (optional) part of the "string" is the "length modification
42 character". This character is the letter "l". This letter can only
42 be used in conjunction with the "conversion characters": d, u, o, x
43B:735
41Frame 735 T
42 * Printf Continued *
42
42 The last part of the "string" is the "conversion character". The
42 "conversion character" can be one of 9 different characters.
42
42 d = signed decimal notation
42 u = unsigned decimal notation
42 o = unsigned octal notation
42 x = unsigned hexadecimal notation
42
42 f = float or double decimal notation (precision default = 6)
42 e = float or double scientific notation (precision default = 6)
42 q = float or double using the shorter of e or f above
42
42 s = string
42 c = character
43B:740
41Frame 740 T
42 * Printf Continued *
42
42 Let's look at a couple of examples involving the "printf" function call.
42
42 printf("%d %f",int_var,float_var);

```

42
 42 The above call will print to standard output (users terminal screen)
 42 two numbers of the types "integer" and "floating point real".
 42
 42 The users printed output numbers would be of the form: 23 45.78
 42
 42 The "printf" function will print the first argument ("int_var") and
 42 then print the second argument ("float_var").
 42
 42 Note: A "white space" or blank character is printed between the argu-
 42 ments since one space appears between the conversion specifications
 42 in the "format control string".
 43B:745
 41Frame 745 T
 42 * Printf Continued *
 42
 42 As another example: printf("\n%6.2f",float_var);
 42
 42 This "printf" call will execute a "new line" and then print a "float-
 42 ing point real" right justified in a field of 6 print positions with
 42 2 digits after the decimal point.
 42
 42 The users output would look something like this: 2561.89
 42
 42 The function "printf" will print the value in "float_var" using the
 42 specified format unless more print positions are needed, in which
 42 case, more print positions will be used.
 43B:750
 41Frame 750 QM
 42Given the function call: printf("%4d %-4.2f %s %c",w,x,y,z);
 42
 42Which of the following variables corresponds to the printed output: HI
 43A w
 43B x
 43C+ y
 43D z
 44 Right. HI is a string.
 44 B:755
 45ABD Wrong. HI is a string, therefore response "C" is the correct answer.
 45 B:755
 41Frame 755 T
 42 *** Lesson Six Summary ***
 42
 42 Well, we have come to the end of lesson six. If you have seen the
 42 four subject topics in this lesson, you should now be ready to take
 42 the final test. If you feel that you don't understand something well
 42 enough to pass the test, please retake the topic that is giving you
 42 problems.
 42
 42 Topic 1 gave a description of the I/O functions "getchar" and "putchar".
 42
 42 Topic 2 gave a description of the I/O function "getline".

42 Topic 3 gave a description of the I/O function "scanf".
42
42 Topic 4 gave a description of the I/O function "printf".
43END
51Frame 900 TT TEST OVER LESSON 6
52 Welcome to the final test of lesson six. This test consists of seven
52 questions over material presented in the previous four topic areas.
52
52 In order to successfully complete this lesson, you must achieve a
52 minimum score of 71.4% (five out of seven questions correct).
52
52 If you miss a question, the correct answer will not be shown. It is
52 up to you to research the correct answer.
52
52 Well, enough said. Let's get on with it. Good luck!
53B:905
51Frame 905 QF
521. The "getchar" function is used to read one character at a time from
52standard input to the executing C program. (True or False)
53Y
54 Right. (1,110)
54 B:910
55 Wrong. (1,110)
55 B:910
51Frame 910 QM
522. Which of the following is "not" a correct use the "putchar" function?
53A putchar(c); Where c is any character variable.
53B+ putchar(*c); Where *c is a pointer to any character array.
53C putchar('c'); Where 'c' is any character constant.
53D putchar('\c'); Where \c is any C escape sequence.
54 Right. (1,135)
54 B:915
55ACD Wrong. (1,135)
55 B:915
55E "E" was not one of your choices.
55 B:910
51Frame 915 QM
523. Given the function call statement: n = getline(input_line,80);
52
52Which of the following is "not" true.
53A "n" must be a variable of type "int".
53B+ "getline" will return two values "n" and "input_line".
53C "input_line" must be a character array.
53D "80" is the maximum input line size.
54 Right. (2,305)
54 B:920
55ACD Wrong. (2,305)
55 B:920
55E "E" was not one of your choices.
55 B:915
51Frame 920 QF
524. Given the function call statement: getline(input_line,35);

52

52The maximum number of characters that will be read by the function "getline"
52is 35. (True or False)

53N

54 Right. (2,310)

54 B:925

55 Wrong. (2,310)

55 B:925

51Frame 925 QP

525. The format of the "scanf" function call is composed of two parts:

52a "format control string" and the "pointer arguments". (True or false)

53Y

54 Right. (3,505)

54 B:930

55 Wrong. (3,505)

55 B:930

51Frame 930 QM

526. Given the function call: scanf("%d%f%s%c",&w,&x,y,&z);

52

52Which of the following variables will contain a number with a decimal point?

53A w

53B+ x

53C y

53D z

54 Right. (3,530)

54 B:935

55ACD Wrong. (3,530)

55 B:935

55E "E" was not one of your choices.

55 B:930

51Frame 935 QP

527. The "printf" function call: printf("\nI Love C"); will execute a "new
52line" and then print the character sequence: I Love C (True or false)

53Y

54 Right. (4,710)

54 B:940

55 Wrong. (4,710)

55 B:940

51Frame 940 T

52 *** End of Lesson/Course Material ***

52

52 This marks the end of lesson number six and hence the end of the
52 course. I hope that the lesson as well as the course was of some
52 benefit to you.

52

52 I hope that you didn't have too much trouble with the material
52 presented in this or any of the lessons in this course. If you
52 did, please voice your comments to your training monitor who will
52 in turn contact the CAI Plans Branch at Keesler AFB, MS.

52

52 Well, let's take a look at how you did with the test ...

53END

File "EXIT"

```

# THE COURSE YOU ARE NOW LEAVING WAS WRITTEN BY CAPT FRANK DEMARCO
# IN PARTIAL FULFILLMENT OF HIS MASTERS DEGREE IN INFORMATION SYSTEMS.
#
#
#
# GGGGGG      000000      000000      DDDDDDD      BBBB BBB      YY      YY      EEEEEEEE
# GGGGGGGG     00000000     00000000     DDDDDDDD     BBBB BBB     YY      YY     EEEEEEEE
# GG          00      00      00      00      DD      DD      BB      BB      YY      YY      EE
# GG      GG   00      00      00      00      DD      DD      BBBB BB      YYY      EEEEEEE
# GG      GG   00      00      00      00      DD      DD      BB      BB      YY      EE
# GGGGGGGG     00000000     00000000     DDDDDDDD     BBBB BBB     YY      EEEEEEEE
# GGGGGGGG      000000      000000      DDDDDDD      BBBB BBB      YY      EEEEEEEE
#
#
# FFFFFFFF     000000      RRRRRR      NN      NN      000000      WW      WW      !!
# FFFFFFFF     00000000     RRRRRRR      NNNN      NN      00000000     WW WW WW      !!
# FF          00      00      RR      RR      NN NN NN      00      00      WW WW WW      !!
# FFFFFFFF     00      00      RRRRRR      NN NN NN      00      00      WW WW WW      !!
# FF          00      00      RR      RR      NN NN NN      00      00      WW WW WW      !!
# FF          00000000     RR      RR      NN      NNNN      00000000     WWWWWWWW
# FF          000000      RR      RR      NN      NN      000000      WWWWWW      00
!

```

VITA

Captain Frank W. DeMarco was born on 8 June 1954 in Wheeling, West Virginia. He graduated from St. Johns High School in Bellaire, Ohio, in 1972 and entered the Air Force at the age of eighteen. He was honorably discharged from the Air Force in 1976 and joined the Ohio Air National Guard. In 1978 he joined the Air Force Reserve Officer Training Corps at Ohio University in Athens, Ohio. He received the degree of Bachelor of Science in Education (Mathematics) in June of 1980. Upon graduation, he received his commission in the USAF. Entering active duty in July 1980 he was assigned to the 3300 Technical Training Wing (TCHTW) at Keesler AFB, Mississippi. His duties while at Keesler included working as a World Wide Military Command and Control System (WWMCCS) mobile training team member and as a course writer for the Computer Assisted Instruction (CAI) Plans Branch of the 3300 TCHTW. In May of 1984 he entered the School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, Ohio.

Permanent address: 212 South 8th Street
Martins Ferry, Ohio
43935

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

AD-A113842

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED		1b. RESTRICTIVE MARKINGS	
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE			
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GCS/MA/85D-2		5. MONITORING ORGANIZATION REPORT NUMBER(S)	
6a. NAME OF PERFORMING ORGANIZATION School of Engineering	6b. OFFICE SYMBOL (If applicable) AFIT/ENC	7a. NAME OF MONITORING ORGANIZATION	
6c. ADDRESS (City, State and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, Ohio 45433		7b. ADDRESS (City, State and ZIP Code)	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Computer Assisted Instruction Plans Branch	8b. OFFICE SYMBOL (If applicable) TTGXZ	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c. ADDRESS (City, State and ZIP Code) CAI Plans Branch 3300 TCHTW/TTGXZ Keesler AFB, MS 39534		10. SOURCE OF FUNDING NOS.	
11. TITLE (Include Security Classification) See Box 19		PROGRAM ELEMENT NO.	PROJECT NO.
		TASK NO.	WORK UNIT NO.
12. PERSONAL AUTHOR(S) Frank W. DeMarco, B.S., Capt, USAF			
13a. TYPE OF REPORT MS Thesis	13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT 226
16. SUPPLEMENTARY NOTATION			
17. COSATI CODES		18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)	
FIELD	GROUP	SUB. GR.	
9	02		
		Computer Applications Teaching Methods Computer Aided Instruction	
19. ABSTRACT (Continue on reverse if necessary and identify by block number)			
Title: COMPUTER ASSISTED INSTRUCTION FOR THE "C" PROGRAMMING LANGUAGE ON THE ZENITH Z-100 MICROCOMPUTER SYSTEM			
Thesis Chairman: Dr. Henry B. Potoczny Professor of Mathematics			
Approved for public release. LAW AFB 180-1. Lynn E. Wolaver 16 JAN 86 Dean for Research and Professional Development Air Force Institute of Technology (AFIT) Wright-Patterson AFB OH 45433			
20. DISTRIBUTION AVAILABILITY OF ABSTRACT UNCLASSIFIED UNLIMITED <input checked="" type="checkbox"/> SAME AS RPT <input type="checkbox"/> DTIC USERS <input type="checkbox"/>		21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. Henry B. Potoczny	22b. TELEPHONE NUMBER (Include Area Code) 513-255-3098	22c. OFFICE SYMBOL AFIT/ENC	

The field known as "computer assisted instruction" or CAI as it is commonly called, has gained considerable interest and support since the advent of the microcomputer. More and more people, including those in supervisory positions are beginning to see the advantages, both cost and time, in having training available in the workplace. This study developed a training package for use on the Zenith Z-100 microcomputer. The package consists of six lessons and three programs. The six lessons cover various topics dealing with the "C" programming language. The objective of these lessons is to present an introduction to the "C" programming language. The three programs are written in the Pascal programming language and are used for the following functions:

1. Provide a means of displaying the lesson material.
2. Provide a means of checking student progress.
3. Provide a means of displaying course statistics.

END

FILMED

3-86

DTIC